



Learning One-Clock Timed Automata*

Jie An^{1(✉)} , Mingshuai Chen^{2,3,4} , Bohua Zhan^{3,4} ,
Naijun Zhan^{3,4(✉)} , and Miaomiao Zhang^{1(✉)}



¹ School of Software Engineering, Tongji University, Shanghai, China
{1510796,miaomiao}@tongji.edu.cn

² Lehrstuhl für Informatik 2, RWTH Aachen University, Aachen, Germany
chenms@cs.rwth-aachen.de

³ State Key Lab. of Computer Science, Institute of Software, CAS, Beijing, China
{bzhan,znj}@ios.ac.cn

⁴ University of Chinese Academy of Sciences, Beijing, China

Abstract. We present an algorithm for active learning of deterministic timed automata with a single clock. The algorithm is within the framework of Angluin’s L^* algorithm and inspired by existing work on the active learning of symbolic automata. Due to the need of guessing for each transition whether it resets the clock, the algorithm is of exponential complexity in the size of the learned automata. Before presenting this algorithm, we propose a simpler version where the teacher is assumed to be *smart* in the sense of being able to provide the reset information. We show that this simpler setting yields a polynomial complexity of the learning process. Both of the algorithms are implemented and evaluated on a collection of randomly generated examples. We furthermore demonstrate the simpler algorithm on the functional specification of the TCP protocol.

Keywords: Automaton learning · Active learning · One-clock timed automata · Timed language · Reset-logical-timed language.

1 Introduction

In her seminal work [10], Angluin introduced the L^* algorithm for learning a regular language from queries and counterexamples within a query-answering framework. The Angluin-style learning therefore is also termed *active learning* or *query learning*, which is distinguished from *passive learning*, i.e., generating a model from a given data set. Following this line of research, an increasing number of efficient active learning methods (cf. [38]) have been proposed to learn, e.g., Mealy machines [34,30], I/O automata [2], register automata [25,1,15], nondeterministic finite automata [12], Büchi automata [19,28], symbolic automata [29,18,11] and Markov decision processes [36], to name just a few. Full-fledged libraries, tools and applications are also available for automata-learning tasks [13,27,20,21].

* This work has been partially funded by NSFC under grant No. 61625206, 61972284, 61732001 and 61872341, by the ERC Advanced Project FRAPPANT under grant No. 787914, and by the CAS Pioneer Hundred Talents Program under grant No. Y9RC585036.

For real-time systems where timing constraints play a key role, however, learning a formal model is much more complicated. As a classical model for real-time systems, timed automata [4] have an infinite set of timed actions. This yields a fundamental difference to finite automata featuring finite alphabets. Moreover, it is difficult to detect resets of clock variables from observable behaviors of the system. This makes learning formal models of timed systems a challenging yet interesting problem.

Various attempts have been carried out in the literature on learning timed models, which can be classified into two tracks. The first track pursues active learning methods, e.g. [22] for learning event-recording automata (ERA) [5] and [9] for learning real-time automata (RTA) [17]. ERA are time automata where, for every untimed action a , a clock is used to record the time of the last occurrence of a . The underlying learning algorithm [22], however, is prohibitively complex due to too many degrees of freedom and multiple clocks for recording events. RTA are a class of special timed automata with one clock to record the execution time of each action by resetting at the starting. The other track pursues passive learning. In [42,41], an algorithm was proposed to learn deterministic RTA. The basic idea is that the learner organizes a tree sketching traces of the data set while merging nodes of the tree following a certain heuristic function. A passive learning algorithm for timed automata with one clock was further proposed in [39,40]. A common weakness of passive learning methods is that the generated model merely accepts all positive traces while it rejects all negative ones for the given set of traces, without guaranteeing that it is a correct model of the target system. A theoretical result was established in [40] showing it is possible to obtain the target system by continuously enriching the data set, however the number of iterations is unknown. In addition, the passive learning methods cited above concern only discrete-time semantics of the underlying timed models, i.e., the clock takes values from non-negative integers. We furthermore refer the readers to [14,32] for learning specialized forms of practical timed systems in a passive manner, [37] for passively learning timed automata using genetic programming which scales to automata of large sizes, [33] for learning probabilistic real-time automata incorporating clustering techniques in machine learning, and [36] for L^* -based learning of Markov decision processes with testing and sampling.

In this paper, we present the first active learning method for deterministic one-clock timed automata (DOTAs) under continuous-time semantics¹. Such timed automata provide simple models while preserving adequate expressiveness, and therefore have been widely used in practical real-time systems [35,3,16]. We present our approach in two steps. First, we describe a simpler algorithm, under the assumption that the teacher is *smart* in the sense of being able to provide information about clock resets in membership and equivalence queries. The basic idea is as follows. We define the *reset-logical-timed language* of a DOTA and show that the timed languages of two DOTAs are equivalent if their reset-logical-timed languages are equivalent, which reduces the learning problem to that of learning a reset-logical-timed language. Then we show how to learn the reset-logical-timed language following Maler and D'Antoni's learning algorithms for symbolic automata [29,18]. We claim the correctness, termination and polynomial complexity of this learning algorithm. Next, we extend this algorithm to the case of a normal teacher. The main difference is that the learner now needs to *guess* the reset

¹ The proposed learning method applies trivially to discrete-time semantics too.

information on transitions discovered in the observation table. Due to these guesses, the latter algorithm features exponential complexity in the size of the learned automata. The proposed learning methods are implemented and evaluated on randomly generated examples. We also demonstrate the simpler, polynomial algorithm on a practical case study concerning the functional specification of the TCP protocol. Detailed proofs for theorems and lemmas in this paper can be found in Appendix A of the full version [7].

In what follows, Sect. 2 provides preliminary definitions on one-clock timed automata. The learning algorithm with a smart teacher is presented and analyzed in Sect. 3. We then present the situation with a normal teacher in Sect. 4. The experimental results are reported in Sect. 5. Finally, Sect. 6 concludes this paper.

2 Preliminaries

Let $\mathbb{R}_{\geq 0}$ and \mathbb{N} be the set of non-negative reals and natural numbers, respectively, and \mathbb{B} the Boolean set. We use \top to stand for true and \perp for false. The projection of an n -tuple \mathbf{x} onto its first two components is denoted by $\Pi_{\{1,2\}} \mathbf{x}$, which extends to a sequence of tuples as $\Pi_{\{1,2\}}(\mathbf{x}_1, \dots, \mathbf{x}_k) = (\Pi_{\{1,2\}} \mathbf{x}_1, \dots, \Pi_{\{1,2\}} \mathbf{x}_k)$.

Timed automata [4], a kind of finite automata extended with a finite set of real-valued clocks, are widely used to model real-time systems. In this paper, we consider a subclass of timed automata with a single clock, termed *one-clock timed automata* (OTAs). Let c be the clock variable, denote by Φ_c the set of clock constraints of the form $\phi ::= \top \mid c \bowtie m \mid \phi \wedge \phi$, where $m \in \mathbb{N}$ and $\bowtie \in \{=, <, >, \leq, \geq\}$.

Definition 1 (One-clock timed automata). A *one-clock timed automaton* $\mathcal{A} = (\Sigma, Q, q_0, F, c, \Delta)$, where Σ is a finite set of actions, called the *alphabet*; Q is a finite set of locations; $q_0 \in Q$ is the initial location; $F \subseteq Q$ is a set of accepting locations; c is the unique clock; and $\Delta \subseteq Q \times \Sigma \times \Phi_c \times \mathbb{B} \times Q$ is a finite set of transitions.

A transition $\delta = (q, \sigma, \phi, b, q')$ allows a jump from the *source location* q to the *target location* q' by performing the action $\sigma \in \Sigma$ if the constraint $\phi \in \Phi_c$ is satisfied. Meanwhile, clock c is reset to zero if $b = \top$, and remains unchanged otherwise.

A *clock valuation* is a function $\nu: c \mapsto \mathbb{R}_{\geq 0}$ that assigns a non-negative real number to the clock. For $t \in \mathbb{R}_{\geq 0}$, let $\nu + t$ be the clock valuation with $(\nu + t)(c) = \nu(c) + t$. According to the definitions of clock valuation and clock constraint, a transition *guard* can be represented as an interval whose endpoints are in $\mathbb{N} \cup \{\infty\}$. For example, $\phi_1: c < 5 \wedge c \geq 3$ is represented as $[3, 5)$, $\phi_2: c = 6$ as $[6, 6]$, and $\phi_3: \top$ as $[0, \infty)$. We will use the inequality- and interval-representation interchangeably in this paper.

A *state* s of \mathcal{A} is a pair (q, ν) , where $q \in Q$ and ν is a clock valuation. A *run* ρ of \mathcal{A} is a finite sequence $\rho = (q_0, \nu_0) \xrightarrow{t_1, \sigma_1} (q_1, \nu_1) \xrightarrow{t_2, \sigma_2} \dots \xrightarrow{t_n, \sigma_n} (q_n, \nu_n)$, where $\nu_0(c) = 0$, $t_i \in \mathbb{R}_{\geq 0}$ stands for the time delay spending on q_{i-1} before $\delta_i = (q_{i-1}, \sigma_i, \phi_i, b_i, q_i) \in \Delta$ is taken, only if (1) $\nu_{i-1} + t_i$ satisfies ϕ_i , (2) $\nu_i(c) = \nu_{i-1}(c) + t_i$ if $b_i = \perp$, otherwise $\nu_i(c) = 0$, for all $1 \leq i \leq n$. A run ρ is *accepting* if $q_n \in F$.

The *trace* of a run ρ is a timed word, denoted by $\text{trace}(\rho)$. $\text{trace}(\rho) = \epsilon$ if $\rho = (q_0, \nu_0)$, and $\text{trace}(\rho) = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n)$ if $\rho = (q_0, \nu_0) \xrightarrow{t_1, \sigma_1} (q_1, \nu_1) \xrightarrow{t_2, \sigma_2} \dots \xrightarrow{t_n, \sigma_n} (q_n, \nu_n)$. Since t_i is the time delay on q_{i-1} , for $1 \leq i \leq n$, such a timed

word is also called *delay-timed word*. The corresponding *reset-delay-timed word* can be defined as $\text{trace}_r(\rho) = (\sigma_1, t_1, b_1)(\sigma_2, t_2, b_2) \cdots (\sigma_n, t_n, b_n)$, where b_i is the reset indicator for δ_i , for $1 \leq i \leq n$. If ρ is an accepting run of \mathcal{A} , $\text{trace}(\rho)$ is called an *accepting timed word*. The *recognized timed language* of \mathcal{A} is the set of accepting delay-timed words, i.e., $\mathcal{L}(\mathcal{A}) = \{\text{trace}(\rho) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$. The *recognized reset-timed language* $\mathcal{L}_r(\mathcal{A})$ is defined as $\{\text{trace}_r(\rho) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$.

The delay-timed word $\omega = (\sigma_1, t_1)(\sigma_2, t_2) \cdots (\sigma_n, t_n)$ is observed outside, from the view of the global clock. On the other hand, the behavior can also be observed inside, from the view of the local clock. This results in a *logical-timed word* of the form $\gamma = (\sigma_1, \mu_1)(\sigma_2, \mu_2) \cdots (\sigma_n, \mu_n)$ with $\mu_i = t_i$ if $i = 1 \vee b_{i-1} = \top$ and $\mu_i = \mu_{i-1} + t_i$ otherwise. We will denote the mapping from delay-timed words to logical-timed words above by Γ .

Similarly, we introduce *reset-logical-timed word* $\gamma_r = (\sigma_1, \mu_1, b_1)(\sigma_2, \mu_2, b_2) \cdots (\sigma_n, \mu_n, b_n)$ as the counterpart of $\omega_r = (\sigma_1, t_1, b_1)(\sigma_2, t_2, b_2) \cdots (\sigma_n, t_n, b_n)$ in terms of the local clock. Without any substantial change, we can extend the mapping Γ to map reset-delay-timed words to reset-logical-timed words. The *recognized logical-timed language* of \mathcal{A} is given as $L(\mathcal{A}) = \{\Gamma(\text{trace}(\rho)) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$, and the *recognized reset-logical-timed language* of \mathcal{A} as $L_r(\mathcal{A}) = \{\Gamma(\text{trace}_r(\rho)) \mid \rho \text{ is an accepting run of } \mathcal{A}\}$.

An OTA is a *deterministic one-clock timed automaton* (DOTA) if there is at most one run for a given delay-timed word. In other words, for any location $q \in Q$ and action $\sigma \in \Sigma$, the guards of transitions outgoing from q labelled with σ are disjoint subsets of $\mathbb{R}_{\geq 0}$. We say a DOTA is *complete* if for any of its location $q \in Q$ and action $\sigma \in \Sigma$, the corresponding guards form a partition of $\mathbb{R}_{\geq 0}$. This means any given delay-timed word has exactly one run. Any DOTA \mathcal{A} can be transformed into a complete DOTA (referred to as COTA) \mathbb{A} accepting the same timed language as follows: (1) Augment Q with a “sink” location q_s which is not an accepting location; (2) For every $q \in Q$ and $\sigma \in \Sigma$, if there is no outgoing transition from q labelled with σ , introduce a (resetting) transition from q to q_s with label σ and guard $[0, \infty)$; (3) Otherwise, let S be the subset of $\mathbb{R}_{\geq 0}$ not covered by the guards of transitions from q with label σ . Write S as a union of intervals I_1, \dots, I_k in a minimal way, then introduce a (resetting) transition from q to q_s with label σ and guard I_j for each $1 \leq j \leq k$.

From now on, we therefore assume that we are working with COTAs.

Example 1. Fig. 1 depicts the transformation of a DOTA \mathcal{A} (left part) into a COTA \mathbb{A} (right part). First, a non-accepting “sink” location q_s is introduced. Second, we introduce three fresh transitions (marked in blue) from q_1 to q_s as well as transitions from q_s to itself. At last, for location q_0 and label a , the existing guards cover $(1, 3)$, with complement $[0, 1] \cup [3, \infty)$. Hence, we introduce transitions $(q_0, a, [0, 1], \top, q_s)$ and $(q_0, a, [3, \infty), \top, q_s)$. Two fresh transitions from q_1 to q_s are introduced similarly.

3 Learning from a Smart Teacher

In this section, we consider the case of learning a COTA \mathbb{A} with a smart teacher. Our learning algorithm relies on the following reduction of the equivalence over timed languages to that of reset-logical timed languages.

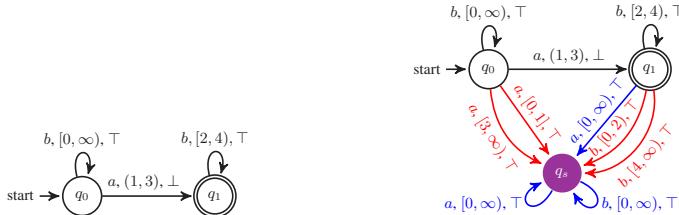


Fig. 1: A DOTA \mathcal{A} on the left and the corresponding COTA \mathbb{A} on the right. The initial location is indicated by ‘start’ and an accepting location is doubly circled.

Theorem 1. *Given two DOTAs \mathcal{A} and \mathcal{B} , if $L_r(\mathcal{A}) = L_r(\mathcal{B})$, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

Theorem 1 assures that $L_r(\mathcal{H}) = L_r(\mathbb{A})$ implies $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathbb{A})$, that is, to construct a COTA \mathbb{A} that recognizes a target timed language $\mathcal{L} = \mathcal{L}(\mathbb{A})$, it suffices to learn a hypothesis \mathcal{H} which recognizes the same reset-logical timed language. For equivalence queries, instead of checking directly whether $L_r(\mathcal{H}) = L_r(\mathbb{A})$, the contraposition of Theorem 1 guarantees that we can perform equivalence queries over their timed counterparts: if $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathbb{A})$, then \mathcal{H} recognizes the target language already; otherwise, a counterexample making $\mathcal{L}(\mathcal{H}) \neq \mathcal{L}(\mathbb{A})$ yields an evidence also for $L_r(\mathcal{H}) \neq L_r(\mathbb{A})$.

We now describe the behavior of the teacher who keeps an automaton \mathbb{A} to be learnt, while providing knowledge about the automaton by answering membership and equivalence queries through an oracle she maintains. For the membership query, the teacher receives a logical-timed word γ and returns whether γ is in $L(\mathbb{A})$. In addition, she is smart enough to return the reset-logical-timed word γ_r that corresponds to γ (the exact correspondence is described in Sect. 3.1). For the equivalence query, the teacher is given a hypothesis \mathcal{H} and decides whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathbb{A})$. If not, she is smart enough to return a reset-delayed-timed word ω_r as a counterexample. The usual case where a teacher can deal with only standard delay-timed words will be discussed in Sect. 4.

Remark 1. The assumption that the teacher can respond with timed words coupled with reset information is reasonable, in the sense that the learner can always infer and detect the resets of the logical clock by referring to a global clock on the wall, as long as he can observe running states of \mathbb{A} , i.e., observing the clock valuation of the system whenever an event happens therein. This conforms with the idea of combining automata learning with white-box techniques, as exploited in [24], providing that in many application scenarios source code is available for the analysis.

In what follows, we elaborate the learning procedure including membership queries, hypotheses construction, equivalence queries and counterexample processing.

3.1 Membership query

In our setting, the oracle maintained by the smart teacher can be regarded as a COTA \mathbb{A} that recognizes the target timed language \mathcal{L} , and thereby its logical-timed language $L(\mathbb{A})$ and reset-logical-timed counterpart $L_r(\mathbb{A})$. In order to collect enough information

for constructing a hypothesis, the learner makes membership queries as “Is the logical-timed word γ in $L(\mathbb{A})?$ ”. If there does not exist a run ρ such that $\Gamma(\text{trace}(\rho)) = \gamma$, meaning that there is some k such that the run is blocked after the k ’th action (i.e. γ is *invalid*) and hence the teacher gives a negative answer, associated with a reset-logical-timed word γ_r where all b_i ’s with $i > k$ are set to \top ; If there exists a run ρ (which is unique due to the determinacy of \mathbb{A}) that admits γ (i.e., γ is *valid*), the teacher answers “Yes”, if ρ is accepting, or “No” otherwise, while in both cases providing the corresponding reset-logical-timed word γ_r , with $\Pi_{\{1,2\}}\gamma_r = \gamma$.

For the sake of simplicity, we define a function π that maps a logical-timed word to its unique reset-logical-timed counterpart in membership queries. Information gathered from the membership queries is stored in a timed observation table defined as follows.

Definition 2 (Timed observation table). A *timed observation table* for a COTA \mathbb{A} is a 7-tuple $\mathbf{T} = (\Sigma, \Sigma, \Sigma_r, S, R, E, f)$ where Σ is the finite alphabet; $\Sigma = \Sigma \times \mathbb{R}_{\geq 0}$ is the infinite set of logical-timed actions; $\Sigma_r = \Sigma \times \mathbb{R}_{\geq 0} \times \mathbb{B}$ is the infinite set of reset-logical-timed actions; $S, R \subset \Sigma_r^*$ and $E \subset \Sigma^*$ are finite sets of words, where S is called the set of prefixes, R the boundary, and E the set of suffixes. Specifically,

- S and R are disjoint, i.e., $S \cup R = S \uplus R$;
- The empty word is by default both a prefix and a suffix, i.e., $\epsilon \in E$ and $\epsilon \in S$;
- $f: (S \cup R) \cdot E \mapsto \{-, +\}$ is a classification function such that for a reset-logical-timed word γ_r , $\gamma_r \cdot e \in (S \cup R) \cdot E$, $f(\gamma_r \cdot e) = -$ if $\Pi_{\{1,2\}}\gamma_r \cdot e$ is invalid, otherwise if $\Pi_{\{1,2\}}\gamma_r \cdot e \notin L(\mathbb{A})$, $f(\gamma_r \cdot e) = -$, and $f(\gamma_r \cdot e) = +$ if $\Pi_{\{1,2\}}\gamma_r \cdot e \in L(\mathbb{A})$.

Given a table \mathbf{T} , we define $\text{row}: S \cup R \mapsto (E \mapsto \{+, -\})$ as a function mapping each $\gamma_r \in S \cup R$ to a vector indexed by $e \in E$, each of whose components is defined as $f(\gamma_r \cdot e)$, denoting a potential location.

Before constructing a hypothesis \mathcal{H} based on the timed observation table \mathbf{T} , the learner has to ensure that \mathbf{T} satisfies the following conditions:

- *Reduced*: $\forall s, s' \in S: s \neq s'$ implies $\text{row}(s) \neq \text{row}(s')$;
- *Closed*: $\forall r \in R, \exists s \in S: \text{row}(s) = \text{row}(r)$;
- *Consistent*: $\forall \gamma_r, \gamma_r' \in S \cup R, \text{row}(\gamma_r) = \text{row}(\gamma_r')$ implies $\text{row}(\gamma_r \cdot \sigma_r) = \text{row}(\gamma_r' \cdot \sigma_r')$, for all $\sigma_r, \sigma_r' \in \Sigma_r$ satisfying $\gamma_r \cdot \sigma_r, \gamma_r' \cdot \sigma_r' \in S \cup R$ and $\Pi_{\{1,2\}}\sigma_r = \Pi_{\{1,2\}}\sigma_r'$;
- *Evidence-closed*: $\forall s \in S$ and $\forall e \in E$, the reset-logical-timed word $\pi(\Pi_{\{1,2\}}s \cdot e)$ belongs to $S \cup R$;
- *Prefix-closed*: $S \cup R$ is prefix-closed.

A timed observation table \mathbf{T} is *prepared* if it satisfies the above five conditions. To get the table prepared, the learner can perform the following operations:

Making \mathbf{T} closed. If \mathbf{T} is not closed, there exists $r \in R$ such that for all $s \in S$ $\text{row}(r) \neq \text{row}(s)$. The learner thus can move such r from R to S . Moreover, each reset-logical-timed word $\pi(\Pi_{\{1,2\}}r \cdot \sigma)$ needs to be added to R , where $\sigma = (\sigma, 0)$ for all $\sigma \in \Sigma$. Such an operation is important since it guarantees that at every location all actions in Σ are enabled, while specifying a clock valuation of these actions, despite that some invalid logical-timed words might be involved. Particularly, giving a bottom value 0 as the clock valuation satisfies the precondition of the partition functions that will be described in Sect. 3.2.

Making \mathbf{T} consistent. If \mathbf{T} is not consistent, one inconsistency is resolved by adding $\sigma \cdot e$ to \mathbf{E} , where σ and e can be determined as follows. T being inconsistent implies that there exist two reset-logical-timed words $\gamma_r, \gamma_r' \in \mathbf{S} \cup \mathbf{R}$ at least, such that $\gamma_r \cdot \sigma_r, \gamma_r' \cdot \sigma_r' \in \mathbf{S} \cup \mathbf{R}$ and $\Pi_{\{1,2\}}\sigma_r = \Pi_{\{1,2\}}\sigma_r'$ for some $\sigma_r, \sigma_r' \in \Sigma_r$, with $row(\gamma_r) = row(\gamma_r')$ but $row(\gamma_r \cdot \sigma_r) \neq row(\gamma_r' \cdot \sigma_r')$. So, let $\sigma = \Pi_{\{1,2\}}\sigma_r = \Pi_{\{1,2\}}\sigma_r'$ and $e \in \mathbf{E}$ such that $f(\gamma_r \cdot \sigma_r \cdot e) \neq f(\gamma_r' \cdot \sigma_r' \cdot e)$. Thereafter, the learner fills the table by making membership queries. Note that this operation keeps the set \mathbf{E} of suffixes being a set of logical-timed words.

Making \mathbf{T} evidence-closed. If \mathbf{T} is not evidence-closed, then the learner needs to add all prefixes of $\pi(\Pi_{\{1,2\}}s \cdot e)$ to \mathbf{R} for every $s \in \mathbf{S}$ and $e \in \mathbf{E}$, except those already in $\mathbf{S} \cup \mathbf{R}$. Similarly, the learner needs to fill the table through membership queries.

The condition that a timed observation table \mathbf{T} is reduced and prefix-closed is inherently preserved by the aforementioned operations, together with the counterexample processing described later in Sect. 3.3. Furthermore, a table may need several rounds of these operations before being prepared (cf. Algorithm 1), since certain conditions may be violated by different, interleaved operations.

3.2 Hypothesis construction

As soon as the timed observation table \mathbf{T} is prepared, a hypothesis can be constructed in two steps, i.e., the learner first builds a DFA \mathbf{M} based on the information in \mathbf{T} , and then transforms \mathbf{M} to a hypothesis \mathcal{H} , which will later be shown as a COTA.

Given a prepared timed observation table $\mathbf{T} = (\Sigma, \Sigma_r, \Sigma_r^*, \mathbf{S}, \mathbf{R}, \mathbf{E}, f)$, a DFA $\mathbf{M} = (Q_M, \Sigma_M, \Delta_M, q_M^0, F_M)$ can be built as follows:

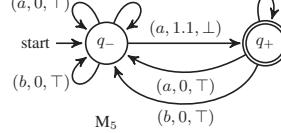
- the finite set of locations $Q_M = \{q_{row(s)} \mid s \in \mathbf{S}\}$;
- the initial location $q_M^0 = q_{row(e)}$ for $e \in \mathbf{S}$;
- the set of accepting locations $F_M = \{q_{row(s)} \mid f(s \cdot \epsilon) = + \text{ for } s \in \mathbf{S} \text{ and } \epsilon \in \mathbf{E}\}$;
- the finite alphabet $\Sigma_M = \{\sigma_r \in \Sigma_r \mid \gamma_r \cdot \sigma_r \in \mathbf{S} \cup \mathbf{R} \text{ for } \gamma_r \in \Sigma_r^*\}$;
- the finite set of transitions $\Delta_M = \{(q_{row(\gamma_r)}, \sigma_r, q_{row(\gamma_r \cdot \sigma_r)}) \mid \gamma_r \cdot \sigma_r \in \mathbf{S} \cup \mathbf{R} \text{ for } \gamma_r \in \Sigma_r^* \text{ and } \sigma_r \in \Sigma_r\}$.

The constructed DFA \mathbf{M} is compatible with the timed observation table \mathbf{T} in the sense captured by the following lemma.

Lemma 1. *For a prepared timed observation table $\mathbf{T} = (\Sigma, \Sigma_r, \Sigma_r^*, \mathbf{S}, \mathbf{R}, \mathbf{E}, f)$, for every $\gamma_r \cdot e \in (\mathbf{S} \cup \mathbf{R}) \cdot \mathbf{E}$, the constructed DFA $\mathbf{M} = (Q_M, \Sigma_M, \Delta_M, q_M^0, F_M)$ accepts $\pi(\Pi_{\{1,2\}}\gamma_r \cdot e)$ if and only if $f(\gamma_r \cdot e) = +$.*

The learner then transforms the DFA \mathbf{M} to a hypothesis $\mathcal{H} = (\Sigma, Q, q_0, F, c, \Delta)$, with $Q = Q_M$, $q_0 = q_M^0$, $F = F_M$, c being the clock and Σ the given alphabet as in \mathbf{T} . The set of transitions Δ in \mathcal{H} can be constructed as follows: For any $q \in Q_M$ and $\sigma \in \Sigma$, let $\Psi_{q,\sigma} = \{\mu \mid (q, (\sigma, \mu, b), q') \in \Delta_M\}$, then applying the partition function $P^c(\cdot)$ (defined below) to $\Psi_{q,\sigma}$ returns k intervals, written as I_1, \dots, I_k , satisfying $\mu_i \in I_i$ for any $1 \leq i \leq k$, where $k = |\Psi_{q,\sigma}|$; consequently, for every $(q, (\sigma, \mu_i, b_i), q') \in \Delta_M$, a fresh transition $\delta_i = (q, \sigma, I_i, b_i, q')$ is added to Δ .

\mathbf{T}_5	ϵ
	ϵ
$(a, 1, \perp)$	+
$(a, 1, \perp)$	-
$(b, 0, \top)$	-
$(a, 1, \perp)(a, 0, \top)$	-
$(a, 1, \perp)(b, 0, \top)$	-
$(a, 1, \perp)(b, 2, \top)$	+
	$(a, 3, \top)$



DFA M_5 has two states, q_- and q_+ . The start state is q_- . Transitions are labeled with clock valuations: $(a, 0, \top)$ from q_- to q_- , $(a, 3, \top)$ from q_- to q_+ , $(b, 2, \top)$ from q_+ to q_+ , $(a, 1, \perp)$ from q_- to q_+ , $(a, 0, \top)$ from q_+ to q_- , and $(b, 0, \top)$ from q_- to q_- .

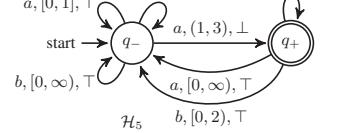


Fig. 2: The prepared timed observation table \mathbf{T}_5 , the corresponding DFA M_5 and hypothesis \mathcal{H}_5 .

Definition 3 (Partition function). Given a list of clock valuations $\ell = \mu_0, \mu_1, \dots, \mu_n$ with $0 = \mu_0 < \mu_1 \dots < \mu_n$, and $\lfloor \mu_i \rfloor \neq \lfloor \mu_j \rfloor$ if $\mu_i, \mu_j \in \mathbb{R}_{\geq 0} \setminus \mathbb{N}$ and $i \neq j$ for all $1 \leq i, j \leq n$, let $\mu_{n+1} = \infty$, then a partition function $P^c(\cdot)$ mapping ℓ to a set of intervals $\{I_0, I_1, \dots, I_n\}$, which is a partition of $\mathbb{R}_{\geq 0}$, is defined as

$$I_i = \begin{cases} [\mu_i, \mu_{i+1}), & \text{if } \mu_i \in \mathbb{N} \wedge \mu_{i+1} \in \mathbb{N}; \\ (\lfloor \mu_i \rfloor, \mu_{i+1}), & \text{if } \mu_i \in \mathbb{R}_{\geq 0} \setminus \mathbb{N} \wedge \mu_{i+1} \in \mathbb{N}; \\ [\mu_i, \lfloor \mu_{i+1} \rfloor], & \text{if } \mu_i \in \mathbb{N} \wedge \mu_{i+1} \in \mathbb{R}_{\geq 0} \setminus \mathbb{N}; \\ (\lfloor \mu_i \rfloor, \lfloor \mu_{i+1} \rfloor], & \text{if } \mu_i \in \mathbb{R}_{\geq 0} \setminus \mathbb{N} \wedge \mu_{i+1} \in \mathbb{R}_{\geq 0} \setminus \mathbb{N}. \end{cases}$$

Remark 2. Definition 3 is adapted from that in [18] by imposing additional assumptions of the list of clock valuations in order to guarantee $\mu_i \in I_i$, for any $0 \leq i \leq n$, due to the underlying continuous-time semantics. Whereas, by \mathbf{T} being prepared and the normalization function described in Sect. 3.3, the set of clock valuations $\Psi_{q,\sigma}$ can be arranged into a list $\ell_{q,\sigma} = \mu_0, \mu_1, \dots, \mu_n$ satisfying such assumptions given in Definition 3 for any $q \in Q_M$ and $\sigma \in \Sigma$.

Example 2. Suppose \mathbb{A} in Fig. 1 recognizes the target timed language. Then the prepared table \mathbf{T}_5 , the corresponding DFA M_5 and hypothesis \mathcal{H}_5 are depicted in Fig. 2. Here, the subscript 5 indicates the fifth iteration of \mathbf{T} (Details concerning the constructions and the entire learning process are enclosed in Appendix B of [7].).

Lemma 2. Given a DFA $M = (Q_M, \Sigma_M, \delta_M, q_M^0, F_M)$, which is generated from a prepared timed observation table \mathbf{T} , the hypothesis $\mathcal{H} = (\Sigma, Q, q_0, F, c, \Delta)$ is transformed from M . For all $\gamma_r \cdot e \in (\mathbf{S} \cup \mathbf{R}) \cdot \mathbf{E}$, \mathcal{H} accepts the reset-logical-timed word $\pi(\Pi_{\{1,2\}} \gamma_r \cdot e)$ iff $f(\gamma_r \cdot e) = +$.

Theorem 2. The hypothesis \mathcal{H} is a COTA.

Given a clock valuation μ , we denote the region containing μ as $\llbracket \mu \rrbracket$, defined as $\llbracket \mu \rrbracket = [\mu, \mu]$ if $\mu \in \mathbb{N}$, and $\llbracket \mu \rrbracket = (\lfloor \mu \rfloor, \lfloor \mu \rfloor + 1)$ otherwise. The following theorem establishes the compatibility of the constructed hypothesis \mathcal{H} with the timed observation table \mathbf{T} .

Theorem 3. For $\gamma_r \cdot e \in (\mathbf{S} \cup \mathbf{R}) \cdot \mathbf{E}$, let $\pi(\Pi_{\{1,2\}} \gamma_r \cdot e) = (\sigma_1, \mu_1, b_1) \dots (\sigma_n, \mu_n, b_n)$. Then for every $\mu'_i \in \llbracket \mu_i \rrbracket$, the hypothesis \mathcal{H} accepts the reset-logical-timed word $\gamma'_r = (\sigma_1, \mu'_1, b_1) \dots (\sigma_n, \mu'_n, b_n)$ if $f(\gamma_r \cdot e) = +$, and cannot accept it if $f(\gamma_r \cdot e) = -$.

3.3 Equivalence query and counterexample processing

Suppose that the teacher knows a COTA \mathbb{A} which recognizes the target timed language \mathcal{L} . Then to answer an equivalence query is to determine whether $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathbb{A})$, which can be divided into two timed language inclusion problems, i.e., whether $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathbb{A})$ and $\mathcal{L}(\mathbb{A}) \subseteq \mathcal{L}(\mathcal{H})$. Most decision procedures for language inclusion proceed by complementation and emptiness checking of the intersection [23]: $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ iff $\mathcal{L}(A) \cap \overline{\mathcal{L}(B)} = \emptyset$. The fact that deterministic timed automata can be complemented [6] enables solving the inclusion problem by checking the emptiness of the resulted product automata $\mathcal{H} \times \mathbb{A}$ and $\overline{\mathcal{H}} \times \mathbb{A}$. The complementation technique, however, does not apply to nondeterministic timed automata even if with only one single clock [4], which we plan to incorporate in our learning framework in future work. We therefore opt for² the alternative method presented by Ouaknine and Worrell in [31] showing that the language inclusion problem of timed automata with one clock (regardless of their determinacy) is decidable by reduction to a reachability problem on an infinite graph. That is, there exists a delay-timed word ω that leads to a *bad configuration* if $\mathcal{L}(\mathcal{H}) \not\subseteq \mathcal{L}(\mathbb{A})$. In detail, the corresponding run ρ of ω ends in an accepting location in \mathcal{H} but the counterpart ρ' of ω in \mathbb{A} is not accepting. Consequently, the teacher can provide the reset-delay-timed word ω_r resulted from ω as a negative counterexample ctx_- . Similarly, a positive counterexample $ctx_+ = (\omega_r, +)$ can be generated if $\mathcal{L}(\mathbb{A}) \not\subseteq \mathcal{L}(\mathcal{H})$. An algorithm elaborating the equivalence query is provided in Appendix C of the full version [7].

When receiving a counterexample $ctx = (\omega_r, +/-)$, the learner first converts it to a reset-logical-timed word $\gamma_r = \Gamma(\omega_r) = (\sigma_1, \mu_1, b_1)(\sigma_2, \mu_2, b_2) \cdots (\sigma_n, \mu_n, b_n)$. By definition, γ_r and ω_r share the same sequence of transitions in \mathbb{A} . Furthermore, by the contraposition of Theorem 1, γ_r is an evidence for $L_r(\mathcal{H}) \neq L_r(\mathbb{A})$ if ω_r is an evidence for $\mathcal{L}(\mathcal{H}) \neq \mathcal{L}(\mathbb{A})$.

Additionally, by the definition of clock constraints Φ_c , at any location, if an action σ is enabled, i.e., its guard is satisfied, w.r.t. the clock value $\mu \in \mathbb{R}_{\geq 0} \setminus \mathbb{N}$, then σ should be enabled w.r.t. any clock value $\lfloor \mu \rfloor + \theta$ at the location, where $\theta \in (0, 1)$. Specifically, only one transition is available for σ at the location on the interval $[\lfloor \mu \rfloor]$, because the target automaton is deterministic. Therefore, in order to avoid unnecessarily distinguishing timed words and violating the assumptions of the list ℓ for the partition function, the learner needs to apply a *normalization function* g to normalize γ_r .

Definition 4 (Normalization). A normalization function g maps a reset-logical-timed word $\gamma_r = (\sigma_1, \mu_1, b_1)(\sigma_2, \mu_2, b_2) \cdots (\sigma_n, \mu_n, b_n)$ to another reset-logical-timed word by resetting any logical clock to its integer part plus a constant fractional part, i.e., $g(\gamma_r) = (\sigma_1, \mu'_1, b_1)(\sigma_2, \mu'_2, b_2) \cdots (\sigma_n, \mu'_n, b_n)$, where $\mu'_i = \mu_i$ if $\mu_i \in \mathbb{N}$, $\mu'_i = \lfloor \mu_i \rfloor + \theta$ for some fixed constant $\theta \in (0, 1)$ otherwise.

We will instantiate $\theta = 0.1$ in what follows. Clearly our approach works for any other θ valued in $(0, 1)$. This *normalization process* guarantees the assumptions needed for Definition 3.

² Remark that the learning complexity (Sect. 3.5) is measured in terms of the number of queries rather than the time complexity of the specific method for checking the equivalence (nor membership). Additionally, the specific method of equivalence checking is not the main concern.

\mathbf{T}_5 / ϵ	\mathbf{T}_6 / ϵ
$\epsilon / -$	$\epsilon / -$
$(a, 1.1, \perp) / +$	$(a, 1.1, \perp) / +$
$(a, 0, T) / -$	$(a, 0, T) / -$
$(b, 0, T) / -$	$(b, 0, T) / -$
$(a, 1.1, \perp)(a, 0, T) / -$	$(a, 1.1, \perp)(a, 0, T) / -$
$(a, 1.1, \perp)(b, 0, T) / -$	$(a, 1.1, \perp)(b, 0, T) / -$
$(a, 1.1, \perp)(b, 2, T) / +$	$(a, 1.1, \perp)(b, 2, T) / +$
$(a, 3, T) / -$	$(a, 3, T) / -$
$\omega_r = (a, 0, T)(a, 1.3, T), -$	$\gamma_r = (a, 0, T)(a, 1.3, T), -$

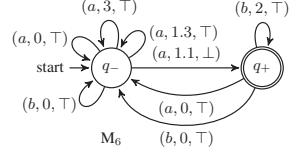


Fig. 3: An illustration of the necessity of normalization by the normalization function.

Algorithm 1: Learning one-clock timed automaton with a smart teacher

```

input : the timed observation table  $\mathbf{T} = (\Sigma, \Sigma_r, S, R, E, f)$ .
output: the hypothesis  $\mathcal{H}$  recognizing the target language  $\mathcal{L}$ .
1  $S \leftarrow \{\epsilon\}; R \leftarrow \{\Gamma(\omega) \mid \omega = (\sigma, 0), \forall \sigma \in \Sigma\}; E \leftarrow \{\epsilon\};$  // initialization
2 fill  $\mathbf{T}$  by membership queries;
3 equivalent  $\leftarrow \perp$ ;
4 while equivalent  $= \perp$  do
5   prepared  $\leftarrow$  is-prepared( $\mathbf{T}$ ); // whether the table is prepared
6   while prepared  $= \perp$  do
7     if  $\mathbf{T}$  is not closed then make_closed( $\mathbf{T}$ );
8     if  $\mathbf{T}$  is not consistent then make_consistent( $\mathbf{T}$ );
9     if  $\mathbf{T}$  is not evidence-closed then make_evidence_closed( $\mathbf{T}$ );
10    prepared  $\leftarrow$  is-prepared( $\mathbf{T}$ );
11    $M \leftarrow$  build_DFA( $\mathbf{T}$ ); // transforming  $\mathbf{T}$  to a DFA M
12    $\mathcal{H} \leftarrow$  build_hypothesis( $M$ ); // constructing a hypothesis  $\mathcal{H}$  from M
13   equivalent, ctx  $\leftarrow$  equivalence_query( $\mathcal{H}$ );
14   if equivalent  $= \perp$  then
15      $\quad$  ctx-processing( $\mathbf{T}, ctx$ ); // counterexample processing
16 return  $\mathcal{H}$ ;

```

Example 3. Consider the prepared table \mathbf{T}_5 in Fig. 3 (as in Fig. 2). When the leaner asks an equivalence query with hypothesis \mathcal{H}_5 , the teacher answers that $\mathcal{L}(\mathcal{H}_5) \neq \mathcal{L}(\mathbb{A})$, where \mathbb{A} in Fig. 1 is the target automaton, and provides a counterexample $(\omega_r, -)$ with $\omega_r = (a, 0, T)(a, 1.3, T)$, which can be transformed to a reset-logical-timed word $\gamma_r = (a, 0, T)(a, 1.3, T)$. If he adds prefixes of γ_r to the table directly, the learner will get a prepared table \mathbf{T}_6 and thus construct a DFA M_6 . Unfortunately, the partition function defined in Definition 3 is not applicable to $(a, 1.3, T)$ and $(a, 1.1, \perp)$ any more. On the other hand, if he adds the prefixes of the normalized reset-logical-timed word, i.e., $\gamma'_r = (a, 0, T)(a, 1.1, T)$, to \mathbf{T}_5 , the learner will then get an inconsistent table whose consistency can be retrieved by the operation of “making \mathbf{T} consistent” as expected.

The following theorem guarantees that the normalized reset-logical-timed word γ'_r is also an evidence for $L_r(\mathcal{H}) \neq L_r(\mathbb{A})$. Therefore, the learner can use it as a counterexample and thus adds all the prefixes of γ'_r to R except those already in $S \cup R$.

Theorem 4. *Given a valid reset-logical-timed word γ_r of \mathbb{A} , its normalization $\gamma'_r = g(\gamma_r)$ shares the same sequence of transitions in \mathbb{A} .*

3.4 Learning algorithm

We present in Algorithm 1 the learning procedure integrating all the previously stated ingredients, including preparing the table, membership and equivalence queries, hypothesis construction and counterexample processing. The learner first initializes the

timed observation table $\mathbf{T} = (\Sigma, \Delta, \Sigma_r, S, R, E, f)$, where $S = \{\epsilon\}$, $E = \{\epsilon\}$, while for every $\sigma \in \Sigma$, he builds a logical-timed word $\gamma = (\sigma, 0)$ and then obtains its reset counterpart $\pi(\gamma) = (\sigma, 0, b)$ by triggering a membership query to the teacher, which is then added to R . Thereafter, the learner can fill the table by additional membership queries. Before constructing a hypothesis, the learner performs several rounds of operations described in Sect. 3.1 until \mathbf{T} is prepared. Then, a hypothesis \mathcal{H} is constructed leveraging an intermediate DFA M and submitted to the teacher for an equivalence query. If the answer is positive, \mathcal{H} recognizes the target language. Otherwise, the learner receives a counterexample ctx and then conducts the counterexample processing to update \mathbf{T} as described in Sect. 3.3. The whole procedure repeats until the teacher gives a positive answer to an equivalence query.

To facilitate the analysis of correctness, termination and complexity of Algorithm 1, we introduce the notion of *symbolic state* that combines each location with its clock regions: a symbolic state of a COTA $\mathbb{A} = (\Sigma, Q, q_0, F, c, \Delta)$ is a pair $(q, [\![\mu]\!])$, where $q \in Q$ and $[\![\mu]\!]$ is a region containing μ . If κ is the maximal constant appearing in the clock constraints of \mathbb{A} , then there exist $2\kappa + 2$ such regions, including $[n, n]$ with $0 \leq n \leq \kappa$, $(n, n+1)$ with $0 \leq n < \kappa$, and (κ, ∞) for each location, so there are a total of $|Q| \times (2\kappa + 2)$ symbolic states. Then the correctness and termination of Algorithm 1 is stated in the following theorem, based on the fact that there is an injection from S (or equivalently, the locations of \mathcal{H}) to symbolic states of \mathbb{A} .

Theorem 5. *Algorithm 1 terminates and returns a COTA \mathcal{H} which recognizes the target timed language \mathcal{L} .*

3.5 Complexity

Given a target timed language \mathcal{L} which is recognized by a COTA \mathbb{A} , let $n = |Q|$ be the number of locations of \mathbb{A} , $m = |\Sigma|$ the size of the alphabet, and κ the maximal constant appearing in the clock constraints of \mathbb{A} . In what follows, we derive the complexity of Algorithm 1 in terms of the number of queries.

By the proof of Theorem 5, \mathcal{H} has at most $n(2\kappa + 2)$ locations (the size of S) distinguished by E . Thus, $|E|$ is at most $n(2\kappa + 2)$ in order to distinguish these locations. Therefore, the number of transitions of \mathcal{H} is bounded by $mn^2(2\kappa + 2)^3$. Furthermore, as every counterexample adds at least one fresh transition to the hypothesis \mathcal{H} , where we consider each interval of the partition corresponds to a transition, this means that the number of counterexamples and equivalence queries is at most $mn^2(2\kappa + 2)^3$.

Now, we consider the number of membership queries, that is, to compute $(|S| + |R|) \times |E|$. Let h be the maximal length of counterexamples returned by the teacher, which is polynomial in the size of \mathbb{A} according to Theorem 5 in [40], bounded by n^2 . There are three cases of extending R by adding fresh rows, namely during the processing of counterexamples, making \mathbf{T} closed, and making \mathbf{T} evidence-closed. The first case adds at most $hmn^2(2\kappa + 2)^3$ rows to R , while the latter two add at most $n(2\kappa + 2) \times m$ and $n^2(2\kappa + 2)^2$, respectively, yielding that the size of R is bounded by $\mathcal{O}(hmn^2\kappa^3)$, where $\mathcal{O}(\cdot)$ is the big Omicron notation. As a consequence, the number of membership queries is bounded by $\mathcal{O}(mn^5\kappa^4)$. So, the total complexity is $\mathcal{O}(mn^5\kappa^4)$.

It is worth noting the above analysis is given in the worst case, where all partitions need to be fully refined. But, in practice we can learn the automaton without refining most partitions, and therefore the number of equivalence and membership queries, as well as the number of locations in the learned automaton are much fewer than the corresponding worst-case bounds. This will be demonstrated by examples in Sect. 5.

3.6 Accelerating Trick

In the timed observation table, the function f maps invalid reset-logical-timed words as well as certain valid ones to “—” when the teacher maintains a COTA \mathbb{A} as the oracle. The learner thus needs multiple rounds of queries to distinguish the “sink” location from other unaccepting locations. If the function f is extended to map invalid reset-logical-timed words to a distinct symbol, say “ \times ”, when we let a DOTA \mathcal{A} be the oracle, then the learner will take much fewer queries. We will later show in the experiments that such a trick significantly accelerates the learning process.

4 Learning from a Normal Teacher

In this section, we consider the problem of learning timed automata with a normal teacher. As before, we assume the timed language to be learned comes from a complete DOTA. For the normal teacher, inputs to membership queries are delay-timed words, and the teacher returns whether the word is in the language (without giving any additional information). Inputs to equivalence queries are candidate DOTAs, and the teacher either answers they are equivalent or provides a delay-timed word as a counterexample.

The algorithm here is based on the procedure in the previous section. We still maintain observation tables where the elements in $S \cup R$ are reset-logical-timed words and the elements in E are logical-timed words. In order to obtain delay-timed words for the membership queries, we need to *guess* clock reset information for transitions in the table. More precisely, in order to convert a logical-timed word to a delay-timed word, it is necessary to know clock reset information for all but the last transition. Hence, it is necessary to guess reset information for each word in $S \cup R$ (since $S \cup R$ is prefix-closed, this is equivalent to guessing reset information for the last transition of each word). Also, for each entry in $(S \cup R) \times E$, it is necessary to guess all but the last transition in E . The algorithm can be thought of as exploring a search tree, where branching is caused by guesses, and successor nodes are constructed by the usual operations of preparing a table and dealing with a counterexample.

The detailed process is given in Algorithm 2. The learner maintains a set of table instances, named *ToExplore*, which contains all table instances that need to be explored.

The initial tables in *ToExplore* are as follows. Each table has $S = E = \{\epsilon\}$. For each $\sigma \in \Sigma$, there is one row in R corresponding to the logical-timed word $\omega = (\sigma, 0)$. It is necessary to guess a reset b for each ω thereby transforming it to a reset-logical-timed word $\gamma_r = (\sigma, 0, b)$. There are $2^{|\Sigma|}$ possible combinations of guesses. These tables are filled by making membership queries (in this case, the membership queries for each table are the same). The resulting $2^{|\Sigma|}$ tables form the initial tables in *ToExplore*.

Algorithm 2: Learning one-clock timed automaton with a normal teacher

```

input : the timed observation table  $\mathbf{T} = (\Sigma, \Sigma_r, S, R, E, f)$ .
output: the hypothesis  $\mathcal{H}$  recognizing the target language  $\mathcal{L}$ .
1  $ToExplore \leftarrow \emptyset; S \leftarrow \{\epsilon\}; R \leftarrow \{\pi(\Gamma(\omega)) \mid \omega = (\sigma, 0), \forall \sigma \in \Sigma\}; E \leftarrow \{\epsilon\};$ 
2  $currentTable \leftarrow (\Sigma, \Sigma_r, S, R, E, f);$ 
3  $tables \leftarrow guess\_and\_fill(currentTable); \quad // guess resets and fill all table instances$ 
4  $ToExplore.insert(tables); \quad // insert table instances tables into ToExplore$ 
5  $currentTable \leftarrow ToExplore.pop(); \quad // pop out head instance as the current table$ 
6  $equivalent \leftarrow \perp;$ 
7 while  $equivalent = \perp$  do
8    $prepared \leftarrow is\_prepared(currentTable); \quad // whether the current table is prepared$ 
9   while  $prepared = \perp$  do
10    if  $currentTable$  is not closed then
11       $tables \leftarrow guess\_and\_make\_closed(currentTable); ToExplore.insert(tables);$ 
12       $currentTable \leftarrow ToExplore.pop();$ 
13    if  $currentTable$  is not consistent then
14       $tables \leftarrow guess\_and\_make\_consistent(currentTable); ToExplore.insert(tables);$ 
15       $currentTable \leftarrow ToExplore.pop();$ 
16    if  $currentTable$  is not evidence-closed then
17       $tables \leftarrow guess\_and\_make\_evidence\_closed(currentTable); ToExplore.insert(tables);$ 
18       $currentTable \leftarrow ToExplore.pop();$ 
19     $prepared \leftarrow is\_prepared(currentTable);$ 
20   $M \leftarrow build\_DFA(currentTable); \quad // transforming currentTable to a DFA M$ 
21   $\mathcal{H} \leftarrow build\_hypothesis(M); \quad // constructing a hypothesis \mathcal{H} from M$ 
22   $equivalent, ctx \leftarrow equivalence\_query(\mathcal{H}); \quad // ctx is a delay-timed word$ 
23  if  $equivalent = \perp$  then
24     $tables \leftarrow guess\_and\_ctx\_processing(currentTable, ctx); \quad // counterexample$ 
    processing
25     $ToExplore.insert(tables);$ 
26     $currentTable \leftarrow ToExplore.pop();$ 
27 return  $\mathcal{H};$ 

```

In each iteration of the algorithm, one table instance is taken out of $ToExplore$. The learner checks whether the table is closed, consistent, and evidence closed. If the table is not closed, i.e. there exists $r \in R$ such that $row(r) \neq row(s)$ for all $s \in S$, the learner moves r from R to S . Then for each $\sigma \in \Sigma$, the element $r \cdot (\sigma, 0)$ is added to R , and a guess has to be made for its reset information. Hence, $2^{|\Sigma|}$ unfilled table instances will be generated. Next, for each entry in the $|\Sigma|$ new rows of R , it is necessary to guess reset information for all but the last transition in $e \in E$. After this guess, it is now possible to fill the table instances by making membership queries with transformed delay-timed words. Hence, there are at most $2^{(\sum_{e_i \in E \setminus \{\epsilon\}} (|e_i|-1) \times |\Sigma|)}$ filled table instances for one unfilled table instance. All filled table instances are inserted into $ToExplore$.

If the table is not consistent, i.e. there exist some $\gamma_r, \gamma'_r \in S \cup R$ and $\sigma_r \in \Sigma_r$ such that $\gamma_r \cdot \sigma_r, \gamma'_r \cdot \sigma_r \in S \cup R$ and $row(\gamma_r) = row(\gamma'_r)$, but $row(\gamma_r \cdot \sigma_r) \neq row(\gamma'_r \cdot \sigma_r)$. Let $e \in E$ be one place where they are different. Then $\sigma_r \cdot e$ needs to be added to E . For each entry in $S \cup R$, all but the last transition in $\sigma_r \cdot e$ need to be guessed, then the table can be filled. $2^{(|\sigma \cdot e| - 1) \times (|S| + |R|)}$ filled table instances will be generated and inserted into $ToExplore$. The operation for making tables evidence-closed is analogous.

Once the current table is prepared, the learner builds a hypothesis \mathcal{H} and makes an equivalence query to the teacher. If the answer is positive, then \mathcal{H} is a COTA which recognizes the target timed language \mathcal{L} ; otherwise, the teacher gives a delay-timed word ω as a counterexample. The learner first finds the longest reset-logical-timed word in

\mathbf{R} which, when converted to a delay-timed word, agrees with a prefix of ω . The remainder of ω , however, needs to be converted to a reset-logical-timed word by guessing reset information. The corresponding prefixes are then added to \mathbf{R} . Hence, at most $2^{|\omega|}$ unfilled table instances are generated. For each unfilled table instance, at most $2^{(\sum_{e_i \in E \setminus \{\epsilon\}} (|e_i| - 1)) \times |\omega|}$ filled tables are produced and inserted into $ToExplore$.

Throughout the learning process, the learner adds a finite number of table instances to $ToExplore$ at every iteration. Hence, the search tree is finite-branching. Moreover, if all guesses are correct, the resulting table instance will be identical to the observation table in the learning process with a smart teacher (apart from the guessing processes, the basic table operations are the same as those in Section 3.1). This means, with an appropriate search order, for example, taking the table instance that requires the least number of guesses in $ToExplore$ at every iteration, the algorithm terminates and returns the same table as in the learning process with a smart teacher, which is a COTA that recognizes the target language \mathcal{L} . In conformity to Theorem 1, the algorithm may terminate even if the corresponding reset-logical-timed languages are not equivalent, while yielding correct COTAs recognizing the same delay-timed language.

Theorem 6. *Algorithm 2 terminates and returns a COTA \mathcal{H} which recognizes the target timed language \mathcal{L} .*

Complexity analysis. If $\mathbf{T} = (\Sigma, \Sigma_r, \mathbf{S}, \mathbf{R}, E, f)$ is the final observation table for the correct candidate COTA, the number of guessed resets in $\mathbf{S} \cup \mathbf{R}$ is $|\mathbf{S}| + |\mathbf{R}|$, and the number of guessed resets for entries in each row of the table is $\sum_{e_i \in E \setminus \{\epsilon\}} (|e_i| - 1)$. Hence, the total number of guessed resets is $(|\mathbf{S}| + |\mathbf{R}|) \times (1 + \sum_{e_i \in E \setminus \{\epsilon\}} (|e_i| - 1))$. Assuming an appropriate search order (for example according to the number of guesses in each table), this yields the number of table instances considered before termination as $\mathcal{O}(2^{(|\mathbf{S}| + |\mathbf{R}|) \times (1 + \sum_{e_i \in E \setminus \{\epsilon\}} (|e_i| - 1))})$.

5 Implementation and Experimental Results

To investigate the efficiency and scalability of the proposed methods, we implemented a prototype³ in PYTHON for learning deterministic one-clock timed automata. The examples include a practical case concerning the functional specification of the TCP protocol [26] and a set of randomly generated DOTAs to be learnt. All of the evaluations have been carried out on a 3.6GHz Intel Core-i7 processor with 8GB RAM running 64-bit Ubuntu 16.04.

Functional specification of the TCP protocol. In [26], a state diagram on page 23 specifies state changes during a TCP connection triggered by causing events while leading to resulting actions. As observed by Ouaknine and Worrell in [31], such a functional specification of the protocol can be represented as a one-clock timed automaton. In our setting, the corresponding DOTA \mathcal{A} to be learnt is configured to have $|Q| = 11$ states with the two CLOSED states collapsed, $|\Sigma| = 10$ after abstracting the causing events and the resulting actions, $|F| = 2$, and $|\Delta| = 19$ with appropriately specified timing constraints including guards and resets. Using the algorithm with the smart teacher, a

³ Available at <https://github.com/Leslieaj/OTALearning>. The evaluated artifact is archived in [8].

Table 1: Experimental results on random examples for the smart teacher situation.

Case ID	$ \Delta _{\text{mean}}$	#Membership			#Equivalence			n_{mean}	t_{mean}
		N_{\min}	N_{mean}	N_{\max}	N_{\min}	N_{mean}	N_{\max}		
4_4_20	16.3	118	245.0	650	20	30.1	42	4.5	24.7
7_2_10	16.9	568	920.8	1393	23	31.3	37	9.1	14.6
7_4_10	25.7	348	921.7	1296	34	50.9	64	9.3	38.0
7_6_10	26.0	351	634.5	1050	35	44.7	70	7.8	49.6
7_4_20	34.3	411	1183.4	1890	52	70.5	93	9.5	101.7
10_4_20	39.1	920	1580.9	2160	61	73.1	88	11.7	186.7
12_4_20	47.6	1090	2731.6	5733	66	97.4	125	16.0	521.8
14_4_20	58.4	1390	2238.6	4430	79	107.7	135	16.0	515.5

Case ID: n_m_κ , consisting of the number of locations, the size of the alphabet and the maximum constant appearing in the clock constraints, respectively, of the corresponding group of \mathcal{A} 's.

$|\Delta|_{\text{mean}}$: the average number of transitions in the corresponding group.

#Membership & #Equivalence: the number of conducted membership and equivalence queries, respectively. N_{\min} : the minimal, N_{mean} : the mean, N_{\max} : the maximum.

n_{mean} : the average number of locations of the learned automata in the corresponding group.

t_{mean} : the average wall-clock time in seconds, including that taken by the learner and the teacher.

correct DOTA \mathcal{H} is learned in 155 seconds after 2600 membership queries and 28 equivalence queries. Specifically, \mathcal{H} has 15 locations excluding a sink location connected by 28 transitions. The introduction of 4 new locations comes from splitting of guards along transitions, which however can be trivially merged back with other locations. The figures depicting \mathcal{A} and \mathcal{H} can be found in Appendix D of [7].

Random examples for a smart teacher. We randomly generated 80 DOTAs in eight groups, with each group having different numbers of locations, size of alphabet, and maximum constant appearing in clock constraints. As shown in Table 1, the proposed learning method succeeds in all cases in identifying a DOTA that recognizes the same timed language. In particular, the number of membership queries and that of equivalence queries appear to grow polynomially with the size of the problem⁴, and are much smaller than the worst-case bounds estimated in Sect. 3.5. Moreover, the learned DOTAs do not have prominent increases in the number of locations (by comparing n_{mean} with the first component of Case IDs). The average wall-clock time including both time taken by the learner and by the teacher is recorded in the last column t_{mean} , of which, however, often over 90% is spent by the teacher for checking equivalences w.r.t. small \mathbf{T} 's while around 50% by the learner for checking the preparedness condition w.r.t. large \mathbf{T} 's.

It is worth noting that all of the results reported above are carried out on an implementation equipped with the *accelerating trick* discussed in Sect. 3.6. We remark that when *dropping* this trick, the average number of membership queries blow up with a factor of 0.83 (min) to 15.02 (max) with 2.16 in average for all the 8 groups, and 0.84 (min) to 1.71 (max) with 1.04 for the average number of equivalence queries, leading to dramatic increases also in the computation time (including that in operating tables).

⁴ An exception w.r.t. the group 7_6_10 is due to relatively simple DOTAs generated occasionally.

Table 2: Experimental results on random examples for the normal teacher situation.

Case ID	$ \Delta _{\text{mean}}$	#Membership			#Equivalence			n_{mean}	t_{mean}	#T _{explored}	#Learned
		N _{min}	N _{mean}	N _{max}	N _{min}	N _{mean}	N _{max}				
3_2_10	4.8	43	83.7	167	5	8.8	14	3.0	0.9	149.1	10/10
4_2_10	6.8	67	134.0	345	6	13.3	24	4.0	7.4	563.0	10/10
5_2_10	8.8	75	223.9	375	9	15.2	24	5.0	35.5	2811.6	10/10
6_2_10	11.9	73	348.3	708	10	16.7	30	5.6	59.8	5077.6	7/10
4_4_20	16.3	231	371.0	564	27	30.9	40	4.0	137.5	8590.0	6/10

#Membership & #Equivalence: the number of conducted membership and equivalence queries with the cached methods, respectively. N_{min}: the minimal, N_{mean}: the mean, N_{max}: the maximum.

#T_{explored}: the average number of the explored table instances.

#Learned: the number of the learnt DOTAs in the group (learnt/total).

The alternative implementation and experimental results without the accelerating trick can also be found in the tool page (under the dev branch).

Random examples for a normal teacher. Due to its high, exponential complexity, the algorithm with a normal teacher failed (out of memory) in identifying DOTAs for almost all the above examples, except 6 cases out of the 10 in group 4_4_20. We therefore randomly generated 40 extra DOTAs of smaller size classified into 4 groups. With the accelerating trick, the learner need not guess the resets in elements of E for an entry in $S \cup R$ if the querying result of the entry is the sink location. We also omitted the checking of the evidence-closed condition, since it may add redundant rows in R , leading to more guesses and thereby a larger search space. The omission does not affect the correctness of the learnt DOTAs. Moreover, as different table instances may generate repeated queries, we cached the results of membership queries and counterexamples, such that the numbers of membership and equivalence queries to the teacher can be significantly reduced. Table 2 shows the performance of the algorithm in this setting. Results without caching are available in the tool page (under the normal branch).

6 Conclusion

We have presented a polynomial active learning method for deterministic one-clock timed automata from a smart teacher who can tell information about clock resets in membership and equivalence queries. Our technique is based on converting the problem to that of learning reset-logical-timed languages. We then extend the method to learning DOTAs from a normal teacher who receives delay-timed words for membership queries, while the learner guesses the reset information in the observation table. We evaluate both algorithms on randomly generated examples and, for the former case, the functional specification of the TCP protocol.

Moving forward, an extension of our active learning method to nondeterministic OTAs and timed automata involving multiple clocks is of particular interest.

Data Availability Statement The datasets generated and/or analyzed during the current study are available in the Figshare repository: <https://doi.org/10.6084/m9.figshare.11545983.v3>.

References

1. Aarts, F., Fiterau-Brosteau, P., Kuppens, H., Vaandrager, F.W.: Learning register automata with fresh value generation. In: Leucker, M., Rueda, C., Valencia, F.D. (eds.) ICTAC 2015. LNCS, vol. 9399, pp. 165–183. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-25150-9_11
2. Aarts, F., Vaandrager, F.W.: Learning I/O automata. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 71–85. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_6
3. Abdullah, J., Dai, G., Mohaqeqi, M., Yi, W.: Schedulability analysis and software synthesis for graph-based task models with resource sharing. In: Proceedings of 24th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2018. pp. 261–270. IEEE Computer Society (2018)
4. Alur, R., Dill, D.L.: A theory of timed automata. *Theor. Comput. Sci.* **126**(2), 183–235 (1994)
5. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.* **211**(1-2), 253–273 (1999)
6. Alur, R., Madhusudan, P.: Decision problems for timed automata: A survey. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 1–24. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_1
7. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata (full version). arXiv:1910.10680 (2019), <https://arxiv.org/abs/1910.10680>
8. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata. Figshare (2020), <https://doi.org/10.6084/m9.figshare.11545983.v3>
9. An, J., Wang, L., Zhan, B., Zhan, N., Zhang, M.: Learning real-time automata. SCIENCE CHINA Information Sciences (2020). <https://doi.org/10.1007/s11432-019-2767-4>, to appear.
10. Angluin, D.: Learning regular sets from queries and counterexamples. *Inf. Comput.* **75**(2), 87–106 (1987)
11. Argyros, G., D’Antoni, L.: The learnability of symbolic automata. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 427–445. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-96145-3_23
12. Bollig, B., Habermehl, P., Kern, C., Leucker, M.: Angluin-style learning of NFA. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009. pp. 1004–1009. AAAI Press (2009)
13. Bollig, B., Katoen, J.P., Kern, C., Leucker, M., Neider, D., Piegdon, D.R.: libalf: The automata learning framework. In: Touili, T., Cook, B., Jackson, P.B. (eds.) CAV 2010. LNCS, vol. 6174, pp. 360–364. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_32
14. Caldwell, B., Cardell-Oliver, R., French, T.: Learning time delay Mealy machines from programmable logic controllers. *IEEE Trans. Automation Science and Engineering* **13**(2), 1155–1164 (2016)
15. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Active learning for extended finite state machines. *Formal Asp. Comput.* **28**(2), 233–263 (2016)
16. Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols. *Communications of the ACM* **24**(8), 533–536 (1981)
17. Dima, C.: Real-time automata. *Journal of Automata, Languages and Combinatorics* **6**(1), 3–23 (2001)
18. Drews, S., D’Antoni, L.: Learning symbolic automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 173–189. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_10

19. Farzan, A., Chen, Y., Clarke, E.M., Tsay, Y., Wang, B.: Extending automated compositional verification to the full class of omega-regular languages. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 2–17. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_2
20. Fiterau-Brostea, P., Janssen, R., Vaandrager, F.W.: Combining model learning and model checking to analyze TCP implementations. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 454–471. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-41540-6_25
21. Fiterau-Brostea, P., Lenaerts, T., Poll, E., de Ruiter, J., Vaandrager, F.W., Verleg, P.: Model learning and model checking of SSH implementations. In: Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, SPIN 2017. pp. 142–151. ACM (2017)
22. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. *Theor. Comput. Sci.* **411**(47), 4029–4054 (2010)
23. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing Company (1979)
24. Howar, F., Jonsson, B., Vaandrager, F.W.: Combining black-box and white-box techniques for learning register automata. In: Steffen, B., Woeginger, G.J. (eds.) Computing and Software Science - State of the Art and Perspectives, LNCS, vol. 10000, pp. 563–588. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91908-9_26
25. Howar, F., Steffen, B., Jonsson, B., Cassel, S.: Inferring canonical register automata. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 251–266. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27940-9_17
26. Information Science Institute, University of Southern California: Transmission control protocol (DARPA internet program protocol specification). <https://www.rfc-editor.org/rfc/rfc793.txt> (1981)
27. Isberner, M., Howar, F., Steffen, B.: The open-source LearnLib - A framework for active automata learning. In: Kroening, D., Pasareanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 487–495. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-21690-4_32
28. Li, Y., Chen, Y., Zhang, L., Liu, D.: A novel learning algorithm for Büchi automata based on family of DFAs and classification trees. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 208–226. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_12
29. Maler, O., Mens, I.: Learning regular languages over large alphabets. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 485–499. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_41
30. Margaria, T., Niese, O., Raffelt, H., Steffen, B.: Efficient test-based model generation for legacy reactive systems. In: Proceedings of the 9th IEEE International High-Level Design Validation and Test Workshop, HLDVT 2004. pp. 95–100. IEEE Computer Society (2004)
31. Ouaknine, J., Worrell, J.: On the language inclusion problem for timed automata: Closing a decidability gap. In: Proceedings of the 19th IEEE Symposium on Logic in Computer Science, LICS 2004. pp. 54–63. IEEE Computer Society (2004)
32. Pastore, F., Micucci, D., Mariani, L.: Timed k-Tail: Automatic inference of timed automata. In: Proceedings of 10th IEEE International Conference on Software Testing, Verification and Validation, ICST 2017. pp. 401–411. IEEE Computer Society (2017)
33. Schmidt, J., Ghorbani, A., Hapfelmeier, A., Kramer, S.: Learning probabilistic real-time automata from multi-attribute event logs. *Intell. Data Anal.* **17**(1), 93–123 (2013)
34. Shahbaz, M., Groz, R.: Inferring Mealy machines. In: Cavalcanti, A., Dams, D. (eds.) FM 2009. LNCS, vol. 5850, pp. 207–222. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05089-3_14

35. Stigge, M., Ekberg, P., Guan, N., Yi, W.: The digraph real-time task model. In: Proceedings of 17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011. pp. 71–80. IEEE Computer Society (2011)
36. Tappler, M., Aichernig, B.K., Bacci, G., Eichlseder, M., Larsen, K.G.: L^* -based learning of Markov decision processes. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 651–669. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-30942-8_38
37. Tappler, M., Aichernig, B.K., Larsen, K.G., Lorber, F.: Time to learn - learning timed automata from tests. In: André, É., Stoelinga, M. (eds.) FORMATS 2019. LNCS, vol. 11750, pp. 216–235. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-03-29662-9_13
38. Vaandrager, F.W.: Model learning. Communications of the ACM **60**(2), 86–95 (2017)
39. Verwer, S., de Weerdt, M., Witteveen, C.: One-clock deterministic timed automata are efficiently identifiable in the limit. In: Didiu, A., Ionescu, A., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 740–751. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00982-2_63
40. Verwer, S., de Weerdt, M., Witteveen, C.: The efficiency of identifying timed automata and the power of clocks. Information and Computation **209**(3), 606–625 (2011)
41. Verwer, S., de Weerdt, M., Witteveen, C.: Efficiently identifying deterministic real-time automata from labeled data. Machine Learning **86**(3), 295–333 (2012)
42. Verwer, S., Weerdt, M.D., Witteveen, C.: An algorithm for learning real-time automata. In: Proceedings of the 18th Annual Machine Learning Conference of Belgium and the Netherlands, Benelearn 2007. pp. 57–64 (2007)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

