Mach Learn
DOI 10.1007/s10994-011-5265-4

Efficiently identifying deterministic real-time automata from labeled data

Sicco Verwer · Mathijs de Weerdt · Cees Witteveen

Received: 22 January 2010 / Accepted: 26 September 2011
© The Author(s) 2011. This article is published with open access at Springerlink.com

Abstract We develop a novel learning algorithm RTI for identifying a *deterministic real-time automaton* (DRTA) from labeled time-stamped event sequences. The RTI algorithm is based on the current state of the art in deterministic finite-state automaton (DFA) identification, called evidence-driven state-merging (EDSM). In addition to having a DFA structure, a DRTA contains time constraints between occurrences of *consecutive* events. Although this seems a small difference, we show that the problem of identifying a DRTA is much more difficult than the problem of identifying a DFA: identifying only the time constraints of a DRTA given its DFA structure is already NP-complete. In spite of this additional complexity, we show that RTI is a correct and complete algorithm that converges efficiently (from polynomial time and data) to the correct DRTA in the limit. To the best of our knowledge, this is the first algorithm that can identify a timed automaton model from time-stamped event sequences.

A straightforward alternative to identifying DRTAs is to identify a DFA that models time *implicitly*, i.e., a DFA that uses different states for different points in time. Such a DFA can be identified by first *sampling* the timed sequences using a fixed frequency, and subsequently applying EDSM to the resulting non-timed event sequences. We evaluate the performance of both RTI and this sampling approach experimentally on artificially generated data. In these

Editor: Nicolo Cesa-Bianchi.

The main part of this research was performed when the first author was a PhD student at Delft University of Technology. It has been supported and funded by the Dutch Ministry of Economical Affairs under the SENTER program.

S. Verwer (✉)
Katholieke Universiteit Leuven, Celestijnenlaan 200a, Box 2402, 3001 Heverlee, Belgium
e-mail: siccoverwer@gmail.com

M. de Weerdt · C. Witteveen
Delft University of Technology, Mekelweg 4, 2826 CD, Delft, The Netherlands

M. de Weerdt
e-mail: m.m.deweerdt@tudelft.nl

C. Witteveen
e-mail: c.witteveen@tudelft.nl

Published online: 13 October 2011

experiments RTI outperforms the sampling approach significantly. Thus, we show that if we obtain data from a real-time system, it is easier to identify a DRTA from this data than to identify an equivalent DFA.

Keywords Timed automata · Real-time automata · Identification in the limit · Supervised learning

1 Introduction

We present a novel method for the automatic identification (learning) of a timed model from positive and negative data. This data consists of sequences of events that could have been generated by a real-time system. A positive sequence characterizes the (correct) behavior of the system, and a negative sequence does not (or characterizes faulty behavior). In order to obtain such data, the data collected from observations needs to be labeled, i.e., an expert needs to decide for some data sequences whether the sequence is an example of the system's behavior or not. From such data, we construct a model that agrees with all the positive examples, and none of the negative ones. Moreover, this model is preferably smallest amongst all possible models that are consistent with the data. We prefer a smaller model due to an important principle in learning theory (or science in general), known as Occam's razor. This states that the simplest explanation (making the least assumptions) for a set of observations is the best one. A smaller model is simpler, and therefore a better explanation for the observed sequences.

Intuitively, we are interested in an algorithm that tries to discover the logical structure underlying the observations. This structure can provide insight into the inner workings of the real-time process. In contrast, many other well-known models that are commonly used to identify the behavior of a system are quite difficult to interpret, for example neural networks or support vector machines, see, e.g., Bishop (2006). Such models are mainly useful for classifying new data or predicting the future behavior of a system. An identified model can in addition be used to analyze properties of the process by applying for instance model checking. The ability to analyze an identified model is one of the main reasons why we are interested in identifying automaton models from observations.

A well-known model for characterizing systems is the *deterministic finite-state automaton* (DFA, see e.g., Sudkamp 2006). A DFA is a language model. Hence, its identification (or inference) problem has been well studied in the grammatical inference field (de la Higuera 2005). Knowing this, we would like to take an established method to identify a DFA and apply it to the observed event sequences. When observing a *real-time* system, however, there often is information in addition to the sequence of symbols: the time at which these symbols occur is also available. By itself, the DFA model is too limited to handle this timed information.

A straightforward way to make use of this timed information is to *sample* the timed data. For instance, an event that occurs 3 seconds after the previous event can be modeled as 3 special time tick symbols followed by the event symbol. Afterwards, we can apply a DFA identification algorithm to the sampled data. The result is a DFA model that models time *implicitly*, i.e., that uses different states to model different points in time. A disadvantage of such an approach is that it results in an *exponential blowup* of both the input data and the resulting automaton size. In this paper, we propose a new algorithm that uses the time information directly in order to produce a timed model.

A well-known timed model is the *timed automaton* (TA) (Alur and Dill 1994). In a TA, each symbol of a sequence occurs at a certain point in time. The state transitions of a TA

contain constraints on the time values of these occurrences relative to previous occurrences. Thus the firing of a transition in a TA depends not only on the type of symbol occurring, but also on the time that has elapsed since some previous symbol occurrence. In this way, TAs can for instance be used to model deadlines in real-time systems. A TA models such timing requirements *explicitly*, i.e., using numbers. Because numbers use a binary representation of time, and states use a unary representation of time, such an explicit representation can result in exponentially more compact models than an implicit representation. Therefore, also the time, space, and data required to identify TAs can be exponentially smaller than the time, space, and data required to identify DFAs.

While the problem of identifying a DFA from a data set has been a well-studied problem in the grammatical inference field, see e.g., de la Higuera (2005), there are very few studies of the identification of a TA from data. The most closely related study deals with the problem of learning event recording automata (ERAs), which is a restricted but still powerful class of TAs (Grinchtein et al. 2006). Unfortunately, the proposed algorithm for the identification of ERAs requires an exponential amount of data in the worst case. We are interested in an algorithm that identifies TAs efficiently, from a small (polynomial) amount of data, and therefore cannot apply this identification algorithm. In fact, in previous work (Verwer et al. 2011), we showed that ERAs are too powerful to be identified efficiently. We should therefore look for a class of TAs more restricted than ERAs. We proved for such a class known as one-clock deterministic timed automata (1-DTAs) that it is efficiently (from polynomial time and data) identifiable in the limit from labeled data. This result came as a surprise because the standard method of converting a deterministic TA into a DFA (the region construction, see Alur and Dill 1994) results in exponentially larger DFAs when applied to this restricted class of TAs. Intuitively, the result states that when identifying a timed system, it is more efficient in terms of both time and data requirements to identify a 1-DTA than to identify a DFA. In this paper, we show that this intuition also holds in practice.

We focus in this paper on identifying a simple type of 1-DTA, known as a deterministic real-time automaton (DRTA, see Sect. 2). A DRTA models only the time constraints between two *consecutive* events, instead of between any pair of events. We restrict ourselves to DRTAs and not to the full class of 1-DTAs because DFA identification already is a difficult problem (NP-complete (Gold 1978) and inapproximable within a polynomial (Pitt and Warmuth 1989)). Hence, it makes sense to first focus on simple extensions of DFAs. In addition, although DRTAs are a restricted type of 1-DTAs, they are still expressive enough for many interesting applications.

We provide an algorithm for identifying DRTAs from labeled data sets (Sect. 4). We call this algorithm RTI, which stands for *real-time identification*. The RTI algorithm is based on the currently best-performing algorithm for the identification of DFAs, called evidence-driven state-merging (EDSM) (Lang et al. 1998). The only difference between DFAs and DRTAs are the time constraints. Although this seems like a small difference, the problem of identifying a DRTA is much more difficult than the problem of identifying a DFA. In this paper, we show that the subproblem of identifying only the time constraints of a DRTA is already NP-complete (Sect. 3). In spite of the complexity of this additional problem, RTI is a correct and complete algorithm that converges efficiently to the correct DRTA in the limit under an appropriate evidence value (Sect. 4.4). To the best of our knowledge, we are the first to develop an efficient algorithm for the identification of a class of TAs.

A big difference between EDSM and RTI is that RTI deals with timed data. Therefore, RTI requires a different evidence value (heuristic) than the one used by EDSM, which is based only on non-timed data. We provide a few values for timed data that can be used in RTI (Sect. 5). We then experimentally compare each of these different evidence values

for RTI with a sampling approach (Sect. 6). The sampling approach first replaces the time values in timed strings by special time tick symbols, and then runs EDSM to obtain a DFA representation of a DRTA. We perform the experiments on a large set of artificial data sets generated from randomly generated DRTAs. The main conclusion of these experiments is that RTI significantly outperforms the sampling approach. In addition, they show that the performance of RTI does not depend on the number of possible time values. In contrast, the performance of the sampling approach degrades quickly when we increase this number. We also compare the performance of the different evidence values we introduced in order to deal with the timed data. Furthermore, we include a simple search wrapper around RTI in our experiments. This search-based version of RTI outperforms the non-search version significantly. Moreover, it scores sufficiently good for real-world problems.

In short, in this paper we show the following:

- We show that identifying DRTAs is *more difficult* than identifying DFAs: the subproblem of identifying only the time constraints is already NP-complete.
- In spite of this result, we show it is still *possible* to efficiently identify DRTAs from labeled data: RTI identifies DRTAs efficiently in the limit when given an appropriate evidence value.
- Furthermore, we show that it is *useful* to identify a DRTA: by identifying a DRTA we obtain a better performance than identifying a DFA from sampled data.
- Finally, we provide useful timed evidence values and show that wrapping a search routine around the RTI algorithm improves its performance significantly.

We conclude this paper with a short overview of related work (Sect. 7) and a discussion of the RTI algorithm (Sect. 8), including possible applications and ideas for future work.

2 Deterministic real-time automata

The following exposition uses basic notation from language, automata, and complexity theory. For an introduction the reader is referred to Sipser (1997). In a real-time system, each occurrence of a symbol (event) is associated with a time value, i.e., its time of occurrence. We model these time values using the *natural numbers* $\mathbb{N}$. This is sufficient because in practice we always deal with a finite precision of time, e.g. milliseconds. Timed automata (Alur and Dill 1994) can be used to accept or generate a sequence $\tau = (a_1, t_1)(a_2, t_2)(a_3, t_3) \cdots (a_n, t_n)$ of symbols $a_i \in \Sigma$ paired with time values $t_i \in \mathbb{N}$, called a *timed string*. The *length* of a timed string τ is the number n of such symbol-time value pairs. Every time value t_i in a timed string represents the time (delay) until the occurrence of symbol a_i since the occurrence of the previous symbol a_{i-1}.

In timed automata, timing conditions are added using a finite number of *clocks* and a *clock guard* for each transition. In this paper, we use a class of timed automata known as *real-time automata* (RTAs) (Dima 2001). An RTA has only one clock that represents the time delay between two *consecutive events*. The clock guards for the transitions are then constraints on this time delay. When trying to identify an RTA from data, one can always determine an upper bound on the possible time delays by taking the maximum observed delay in this data. Therefore, we represent a *delay guard* (constraint) $[n, n']$ by a *closed interval* in $\mathbb{N}$. We say that $[n, n']$ is *satisfied* by a time value $t \in \mathbb{N}$ if $t \in [n, n']$. An RTA is defined as follows:

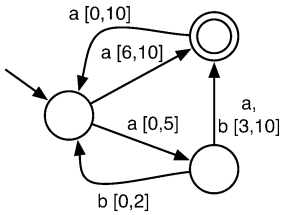

Fig. 1 An example of a DRTA. The leftmost state is the start state, indicated by the sourceless *arrow*. The topmost state is an end state, indicated by the *double circle*. Every state transition contains both a label and a delay guard. Missing transitions lead to a rejecting garbage state

Definition 1 (RTA) A *real-time automaton* (RTA) is a 5-tuple $A = \langle Q, \Sigma, \Delta, q_0, F \rangle$, where Q is a finite set of states, Σ is a finite set of symbols, Δ is a finite set of transitions, q_0 is the start state, and $F \subseteq Q$ is a set of accepting states.

A transition $\delta \in \Delta$ in an RTA is a tuple $\langle q, q', a, [n, n'] \rangle$, where $q, q' \in Q$ are the source and target states, $a \in \Sigma$ is a symbol, and $[n, n']$ is a delay guard.

Due to the complexity of learning non-deterministic automata (see, e.g., de la Higuera 2005), we only consider *deterministic* RTAs (DRTAs). An RTA A is called *deterministic* if A does not contain two transitions with the same symbol, the same source state, and overlapping delay guards. Like timed automata, in DRTAs, it is possible to make time transitions in addition to the normal state transitions used in deterministic finite state automata (DFAs). In other words, during its execution a DRTA can remain in the same state for a while before it generates the next symbol. The time it spends in every state is represented by the time values of a timed string. In a DRTA, a state transition is possible (can fire) only if its delay guard is satisfied by the time spent in the previous state. A transition $\langle q, q', a, [n, n'] \rangle$ of a DRTA is thus interpreted as follows: whenever the automaton is in state q, reading a timed symbol (a, t) such that $t \in [n, n']$, then the DRTA will move to the next state q'.

Example 1 Figure 1 shows an example DRTA. This DRTA accepts and rejects timed strings not only based on their event symbols, but also based on their time values. For instance, it accepts $(a, 4)(b, 3)$ (state sequence: left $\rightarrow$ bottom $\rightarrow$ top) and $(a, 6)(a, 5)(a, 6)$ (left $\rightarrow$ top $\rightarrow$ left $\rightarrow$ top), and rejects $(a, 6)(b, 2)$ (left $\rightarrow$ top $\rightarrow$ reject) and $(a, 5)(a, 5)(a, 6)$ (left $\rightarrow$ bottom $\rightarrow$ top $\rightarrow$ left).

The complete behavior of a DRTA is defined by the *computation* of a DRTA:

Definition 2 (DRTA computation) A finite *computation* of a DRTA $A = \langle Q, \Sigma, \Delta, q_0, F \rangle$ over a (finite) timed string $\tau = (a_1, t_1) \cdots (a_n, t_n)$ is a finite sequence

$$q_0 \xrightarrow{(a_1,t_1)} q_1 \cdots q_{n-1} \xrightarrow{(a_n,t_n)} q_n$$

such that for all $1 \leq i \leq n$, $\langle q_{i-1}, q_i, a_i, [n_i, n_i'] \rangle \in \Delta$, where $t_i \in [n_i, n_i']$. A computation of a DRTA is called *accepting* if $q_n \in F$.

We say that a timed string τ *ends* in a DRTA A in the last state occurring in the computation of A over τ, i.e., τ ends in q_n. A DRTA A accepts a timed string τ if τ ends in a final state, i.e., if $q_n \in F$. The language of a DRTA A, denoted $L(A)$, is the set of timed strings τ that the computation of A over τ is accepting.

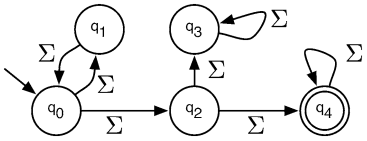

Fig. 2 The incomplete DRTA (without delay guards) resulting from the reduction in the proof of Theorem 1. The alphabet Σ consists of the variables V of the 3-SAT instance

3 Identifying delay guards of a DRTA is NP-complete

Given a timed input sample $S = (S_+, S_-)$, i.e., a pair of sets of positive and negative timed strings, we want to identify the smallest DRTA A that is consistent with S, i.e., the smallest DRTA A such that $S_+ \subseteq L(A)$ and $S_- \subseteq L(A)^c$. This problem adds a difficult subproblem to the DFA identification problem: in addition to identifying the correct DFA structure, the algorithm needs to identify the correct delay guards. Preferably, we would like an algorithm that solves this subproblem optimally. It should then be possible to use this algorithm as a subroutine of a DFA identification algorithm in order to identify DRTAs. Unfortunately, we start this section by proving that identifying only these timed properties of a DRTA is already NP-complete. Thus, we will not be able to solve this subproblem efficiently unless P = NP.

Theorem 1 *Given a timed input sample S and an NFA $A = \langle Q, \Sigma, \Delta, q_0, F\rangle$, the problem of finding for every transition $\delta = \langle q, q', a\rangle \in \Delta$ a delay guard $[n, n']$ such that $\langle Q, \Sigma, \{\langle q, q', a, [n, n']\rangle \mid \langle q, q', a\rangle \in \Delta, n, n' \in \mathbb{N}\}, q_0, F\rangle$ is a DRTA consistent with S is* NP*-complete.*

Proof Our proof is by reduction from 3-SAT (for a definition see e.g. Sipser 1997). We first give the intuition behind this reduction, then we give the formal proof.

The reduction consists of two parts: constructing the NFA A, and constructing the input sample S. The NFA A has a structure that is mostly independent of the 3-SAT instance; only the alphabet Σ contains one symbol a for every variable $a \in V$ in the 3-SAT instance. This structure is depicted in Fig. 2. The main idea of this structure is that the accepting state q_4 can only be reached on an even index. For every clause c in the 3-SAT instance, a positive timed string τ of length 6 is constructed. Hence, every such string τ has 3 opportunities to reach state q_4, once for each literal l in c. Whether they reach state q_4 in one such opportunity depends on the delay guards $[n_1, n_2]$ and $[n_3, n_4]$ of the transitions that have as label the atom a of l:

– if the sign of l is positive, then τ reaches q_4 if $1 \in [n_1, n_2]$ and $1 \in [n_3, n_4]$;
– if the sign of l is negative, then τ reaches q_4 if $0 \in [n_1, n_2]$ and $0 \in [n_3, n_4]$.

For every label $a \in \Sigma$, there is a negative timed string $\tau' \in S_-$ of length 2 that reaches state q_4 only if the delay guards $[n_1, n_2]$ and $[n_3, n_4]$ of the transitions with label a contain both 0 and 1. Thus, the identification of the guards of this NFA A such that all positive examples reach q_4, and none of the negative reach q_4, represents an assignment of truth values to the variables of that 3-SAT instance that satisfies all clauses. We now present the formal proof.

Let $I = (V = \{v_1, \ldots, v_n\}, C = \{c_1, \ldots, c_m\})$ be a 3-SAT instance. We construct the following instance of our delay guard identification problem $(S = (S_+, S_-), A = \langle Q, \Sigma, \Delta, q_0, F\rangle)$ (A is depicted in), where:

- S_+ contains a timed string $\tau_c = (a_1, t_1)(a_1, t_1)(a_2, t_2)(a_2, t_2)(a_3, t_3)(a_3, t_3)$ for every clause $c = \{l_1, l_2, l_3\} \in C$, where for all $1 \leq i \leq 3$, a_i is the atom variable of literal l_i, and $t_i = 1$ if l_i is a positive literal, $t_i = 0$ otherwise;
- S_- contains two timed strings $\tau_v = (a, 0)(a, 1)$ and $\tau'_v = (a, 1)(a, 0)$ for every variable $a \in V$;
- $Q = \{q_0, q_1, q_2, q_3, q_4\}$;
- $\Sigma = V$;
- $\Delta = \bigcup_{a \in V} \left\{ \begin{array}{c} \langle q_0, q_1, a\rangle, \langle q_1, q_0, a\rangle, \langle q_0, q_2, a\rangle, \langle q_2, q_3, a\rangle, \\ \langle q_2, q_4, a\rangle, \langle q_3, q_3, a\rangle, \langle q_4, q_4, a\rangle \end{array} \right\}$;
- $F = \{q_4\}$.

We now claim that there exists a delay guard for every transition $\delta \in \Delta$ such that the resulting DRTA is consistent with S if and only if the original 3-SAT instance is satisfiable.

($\Rightarrow$) Let $m : V \rightarrow \{true, false\}$ be a certificate for the 3-SAT instance I, i.e., setting all variables $v \in V$ to $m(v)$ makes I satisfied. Using m we create the following delay guards for the transitions of A:

- for all $\langle q_0, q_1, a\rangle \in \Delta$ and all $\langle q_2, q_3, a\rangle \in \Delta$ create a guard $[t, t]$, where $t = 0$ if $m(a) = true$, $t = 1$ otherwise;
- for all $\langle q_1, q_0, a\rangle \in \Delta$, all $\langle q_3, q_3, a\rangle \in \Delta$, and all $\langle q_4, q_4, a\rangle \in \Delta$ create a guard $[0, 1]$;
- for all $\langle q_0, q_2, a\rangle \in \Delta$ and all $\langle q_2, q_4, a\rangle \in \Delta$ create a guard $[t', t']$, where $t' = 1$ if $m(a) = true$, $t' = 0$ otherwise.

All the transitions that directly lead to q_4 from the start state q_0 in the DRTA we just constructed have the same delay guard. Because of this, no negative example $\tau_v \in S_-$ can end in the final state q_4. Instead, they end in the non-final state q_3.

For every positive example $\tau_c \in S_+$, there are two transitions τ_c can fire at the start of the computation of A': a transition $\langle q_0, q_1, a\rangle$ to q_1, or a transition $\langle q_0, q_2, a\rangle$ to q_2. It fires a transition to state q_2 if either $m(a) = true$ and the first literal in c is a, or $m(a) = false$ and the first literal in c is $\neg a$. Otherwise τ fires a transition to state q_1.

Suppose the fired transition is to state q_2. Because the delay guard created for all transitions $\langle q_2, q_4, a\rangle$ to state q_4 is equal to the guards created for the transitions $\langle q_0, q_2, a\rangle$, the construction of τ_c guarantees that the next transition it fires is to state q_4. After reaching state q_4, the transitions $\langle q_4, q_4, a\rangle$ with delay guard $[0, 1]$ guarantee that it ends in q_4.

Suppose the fired transition is to state q_1. The next transition it fires is a transition $\langle q_1, q_0, a\rangle$ with delay guard $[0, 1]$ back to q_0. From q_0 it again fires a transition to either q_1 or q_2 depending on the value of $m(a)$ and whether the next literal is positive or not. Since there exists at least one literal in c that is satisfied by m, τ will at some point fire the transition to q_2. Thus, it will at some point end in q_4. Hence, all positive examples $\tau_c \in S_+$ end in q_4. Consequently, it holds that $S_+ \subseteq L(A')$ and $S_- \subseteq L(A')^c$.

($\Leftarrow$) Let A' be a DRTA constructed from A by adding delay guards to transitions such that $S_+ \subseteq L(A')$ and $S_- \subseteq L(A')^c$. Because A accepts all of the positive examples and none of the negative examples, the transitions $\langle q_0, q_2, a, [n_1, n_2]\rangle$ and $\langle q_2, q_4, a, [n_3, n_4]\rangle$ in A' (with the same label) are such that $0 \in [n_1, n_2] \cup [n_3, n_4]$ or $1 \in [n_1, n_2] \cup [n_3, n_4]$, but not $\{0, 1\} \in [n_1, n_2] \cup [n_3, n_4]$. We construct the following solution for the 3-SAT problem: for all $a \in V$, set $m(a) = true$ if $1 \in [n_1, n_2] \cup [n_3, n_4]$, set $m(a) = false$ if $0 \in [n_1, n_2] \cup [n_3, n_4]$. By construction of S_+, and since it holds that $S_+ \subseteq L(A')$, m is a mapping that satisfies at least one literal in every clause of the original 3-SAT problem. Hence, the 3-SAT instance is satisfiable.

The reduction function is clearly polynomial. Checking whether a DRTA is consistent can clearly be done in polynomial time by running the DRTA on every example from the input set. Hence the problem is a member of NP. This completes the proof. □

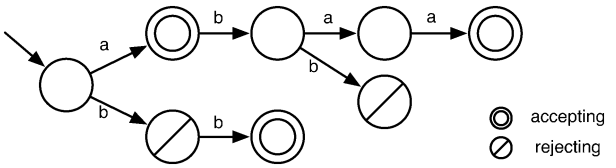

Fig. 3 An augmented prefix tree acceptor for $S = (S_+ = \{a, abaa, bb\}, S_- = \{abb, b\})$. The start state is the state with an arrow pointing to it from nowhere

Despite the above theorem, we still want to identify both the delay guards and the structure of a DRTA. We will not be able to do so efficiently, but we might be able to *converge* efficiently to the correct DRTA when given more and more data, i.e., in the limit. In fact, in previous work (Verwer et al. 2009), we theoretically showed this to be possible. A similar issue holds for the identification of DFAs: even though the DFA identification problem is NP-complete (Gold 1978), there exists a polynomial time algorithm that converges efficiently to the correct DFA (Oncina and Garcia 1992). Therefore, instead of treating the identification of delay guards as a subproblem, we provide a new algorithm for identifying DRTAs that identifies delay guards in a way similar to the way it identifies states and transitions, i.e., such that it converges efficiently to the correct delay guards. We describe this algorithm in the next section.

4 Identifying real-time automata

Our algorithm for identifying DRTAs is called RTI (Algorithm 8, p. 17), which stands for *real-time identification*. The RTI algorithm is based on the evidence-driven state-merging (EDSM) algorithm in a red-blue framework (Algorithm 3), which is one of the most successful algorithms for identifying DFAs (see, e.g., Lang et al. 1998). RTI can perform all of the traditional state-merging routines (Algorithms 4 and 6, pp. 12 and 15). In addition, our DRTA identification algorithm is capable of identifying delay guards by *splitting* transitions into two (Algorithm 5, p. 13). This routine identifies delay guards by first assuming them to contain every time value, and then splitting these constraints into two new non-overlapping constraints for two new transitions.

We start this section with a concise overview of the EDSM algorithm in a red-blue framework (Sect. 4.1). Both the transition-splitting and the modified state-merging routines are then explained in Sect. 4.2. Afterwards, we present the main loop of the RTI algorithm in Sect. 4.3. In Sect. 4.4, we end this section by proving that RTI runs in polynomial time, is correct, is complete, and that it converges efficiently to the correct DRTA.

4.1 State merging

The algorithm we use for the identification of DRTAs is based on to the EDSM algorithm in a red-blue framework for the identification of DFAs (Lang et al. 1998). In this section, we describe the main elements of the original algorithm for DFAs.

An automaton identification process tries to find an automaton A such that its language $L(A)$ is equal to the target language L_t. The identification process should get some form of data as input from which it can identify L_t. We assume it is given a pair of finite sets of positive sample strings $S_+ \subseteq L_t$ and negative sample strings $S_- \subseteq L_t^c$, called the *input*

Algorithm 1 Construct the APTA: APTA

Require: an input sample $S = \{S_+, S_-\}$
Ensure: A is the APTA for S
  A is a DFA containing only a start state q
  for each sample strings $\sigma = a_1, \ldots, a_n$ from S **do**
    state $q' = q$, integer $i = 1$
    while $i \leq n$ **do**
      if A contains no transition δ with q' as source and a_i as symbol **then**
        Add a new state q'' to A.
        Add such a transition δ to A, set its target to be q''.
      end if
      $q' =$ the target of δ, $i = i + 1$
    end while
    if $\sigma \in S_+$ **then**
      Set q' to be an accepting state.
    else
      Set q' to be a rejecting state.
    end if
  end for
  return A

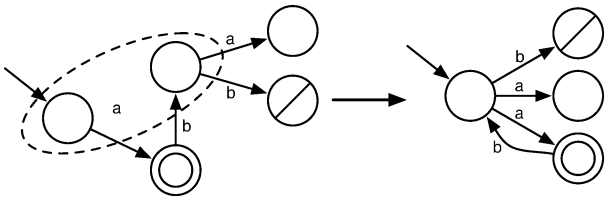

Fig. 4 A merge of two states from the APTA of Fig. 3. *On the left* the original part of the automaton is shown, the states that are to be merged are surrounded by a *dashed ellipse*. *On the right* the result of the merge is shown. This resulting automaton still has to be determinized

sample. The goal is to find the *smallest* DFA that is *consistent* with $S = \{S_+, S_-\}$, i.e., accepting all positive and rejecting all negative strings.

The idea of a state-merging algorithm is to first construct a tree-shaped DFA A from this input, and then to merge the states of A. This DFA A is called an *augmented prefix tree acceptor* (APTA), see Fig. 3. Algorithm 1 shows its construction routine in pseudo code. An APTA A is a DFA that is *consistent* with the input sample S. Furthermore, in an APTA there exists exactly one path from the start state to another state. This implies that the computations of two strings σ and σ' reach the same state q if and only if σ and σ' share the same prefix until they reach q. Hence the name prefix tree. An APTA is called augmented because it contains (is *augmented* with) states that are neither accepting nor rejecting. No execution of any sample string from S ends in such a state. Therefore, it is unknown whether these should be accepting or rejecting. This is determined by merging the states of this APTA and trying to find a DFA that is as small as possible.

A *merge* (see Fig. 4 and Algorithm 2) of two states q and q' combines the states into one: it creates a new state q'' that has the incoming and outgoing transitions of both q and q'. Such a merge is only allowed if the states are *consistent*, i.e. it is not the case that q is accepting while q' is rejecting or vice versa. When a merge introduces a non-deterministic

Algorithm 2 Merging two states: MERGE

Require: an augmented DFA A and two states q, q' from A
Ensure: if q and q' are consistent, then q and q' are merged, A is updated accordingly, and true is returned, false is returned otherwise
  if q is accepting and q' is rejecting or vice versa **then**
    return false
  end if
  Add a new state q'' to A that is neither accepting nor rejecting.
  if q or q' is an accepting state **then**
    Set q'' to be an accepting state.
  end if
  if q or q' is a rejecting state **then**
    Set q'' to be a rejecting state.
  end if
  for each occurrence of q or q' as source or target of transitions A **do**
    Replace the occurrence of q or q' by q''.
  end for
  while A contains a non-deterministic choice with target states q_n and q'_n **do**
    boolean $b = \text{MERGE}(A, q_n, q'_n)$
    if b equals false **then**
      Undo the merge of q with q' and **return** *false*.
    end if
  end while

choice, i.e. q'' is the source of two transitions with the same symbol, the target states of these transitions are merged as well. This is called the *determinization* process (the while loop in Algorithm 2), and is continued until there are no non-deterministic choices left. The result of a merge is a new DFA that is smaller than before, and still consistent with the input sample S. A state-merging algorithm continually applies the state merging process until no more consistent merges are possible.

The red-blue framework follows the state-merging algorithm just described, but in addition maintains a core of red states with a fringe of blue states (see Fig. 5 and Algorithm 3). A red-blue algorithm performs merges only between blue and red states. The new states resulting from these merges are colored red. During a merge, the determinization process can only merge uncolored states with any other state q. The new states resulting from these merges takes the color of q. If no red-blue merge is possible the algorithm changes the color of a blue state into red; we call this changing of color a COLOR operation. The algorithm is guaranteed not to change any of the transitions between red states. The red core of the DFA can be viewed as a part of the DFA that is assumed to be correctly identified.

Note that a red-blue state-merging algorithm is capable of producing any DFA that is consistent with the input sample and smaller than the original APTA. The main goal of a DFA identification algorithm is to find one of the smallest such DFAs. Currently, the most successful method to find this DFA is EDSM (Lang et al. 1998). In EDSM each possible merge is given a score based on the amount of *evidence* in the merges that are performed by the merge and determinization processes. A possible merge gets an evidence score equal to the number of accepting states merged with accepting states plus the number of rejecting states merged with rejecting states. At each iteration of the EDSM algorithm, the merge with the highest evidence value is performed. In the following sections we discuss real-time automata and show how to apply the idea of state merging to these automata.

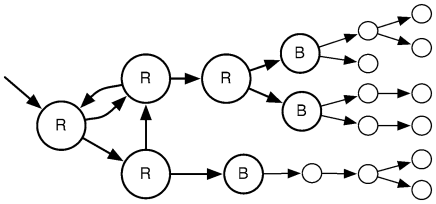

Fig. 5 The red-blue framework. The red states (labeled R) are the identified parts of the automaton. The blue states (labeled B) are the current candidates for merging. The uncolored states are pieces of the APTA

Algorithm 3 State merging in the red-blue framework

Require: an input sample S
Ensure: A is a small DFA that is consistent with S
  $A = \text{APTA}(S)$
  Color the start state q of A red and all the children of q blue.
  while A contains blue nodes **do**
    if A contains a red state r and a blue state b such that $\text{MERGE}(A, r, b)$ equals *true* **then**
      Call $\text{MERGE}(A, r, b)$.
    else
      Change the color of a blue state in A to red.
    end if
    Change the color all uncolored children of red states in A to blue.
  end while
  return A

4.2 Real-time identification

Our DRTA identification algorithm, called RTI, is an EDSM algorithm that uses the red-blue framework. Most of the conventional state-merging routines are modified in order to deal with timed data and DRTAs. Moreover, the algorithm now contains a new routine, called *transition-splitting*, that enables it to identify the delay guards of a DRTA. This routine identifies time constraints by first assuming them to contain every time value, and then splitting these constraints into two new non-overlapping constraints for two new transitions. In previous work (Verwer et al. 2009), we described a one-clock deterministic timed automaton identification algorithm ID_1-DTA that identifies clock guards by simply selecting the smallest consistent one. We use this splitting routine instead of a smallest first order because this routine makes it possible to use an evidence value similar to the one used in the original EDSM algorithm. In fact, it is difficult to define a suitable evidence value for the identification methods in the ID_1DTA algorithm. In the following we discuss all of the changes we made to the original EDSM algorithm.

A timed APTA Like the conventional state-merging algorithm, RTI starts with an augmented prefix tree acceptor (APTA) (A, R), i.e., a prefix tree DRTA A, augmented with a set of rejecting states R. In the conventional APTA, two strings end in the same state only if they are identical. Two timed strings are identical if at every index both their symbols *and* their time values are the same. It rarely occurs that an input sample S contains (prefixes of) timed strings that have exactly the same time values. Hence, were we to construct an APTA in the conventional way, we would with high probability obtain an automaton such

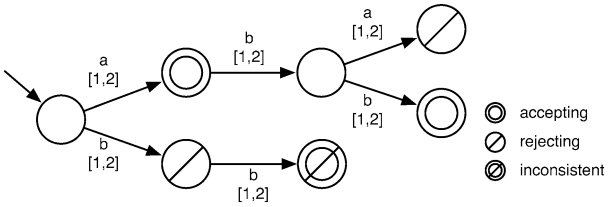

Fig. 6 A timed APTA for the timed input sample: $(S_+ = \{(a,1),(a,1)(b,2)(b,1),(b,2)(b,1)\}, S_- = \{(a,1)(b,1)(a,1),(b,2),(b,1)(b,1)\})$. The minimum delay $t_{\min}$ in the sample is 1, the maximum delay $t_{\max}$ is 2

Algorithm 4 Construct the timed APTA: tapta

Require: an input sample $S = \{S_+, S_-\}$ with alphabet Σ and the global minimum and maximum delay values $t_{\min}$ and $t_{\max}$

Ensure: (A, R) is the timed APTA for S (a DRTA A augmented with a set of rejecting states R)

```
Set A := ⟨Q = {q0}, Σ, Δ = ∅, q0, F = ∅⟩
Set R := ∅
for each timed string τ = (a1, t1), ..., (an, tn) from S do
    Set state q := q0
    for every index 0 < i ≤ n of τ do
        if there exist no ⟨q, q′, ai, [t, t′]⟩ ∈ Δ for any q′ and [t, t′] then
            Add a new state q′ to A
            Add a new transition ⟨q, q′, ai, [tmin, tmax]⟩ to A
        end if
        Set q := q′
    end for
    if τ ∈ S+ then set F = F ∪ {q}
    if τ ∈ S− then set R = R ∪ {q}
end for
return (A, R)
```

that every state will be reached by only a single timed string from S. Starting from such an automaton, the conventional state-merging algorithm could be used to merge the delay guards of transitions into larger delay guards. In other words, we could identify the delay guards in a *bottom-up* way. However, like our ID_1-DTA algorithm, it is very difficult to come up with a good evidence value for such an bottom-up method. The conventional evidence value clearly fails since the determinization routine will only combine states that are reached using the same symbol *and* the same time value(s). In other words, it will usually merge no states at all. Consequently, the evidence value is usually either 1 or 0, and hence it contains little information.

A bottom-up approach for identifying states and transitions of the conventional state-merging algorithm clearly creates some problems for identifying delay guards. Because of this, we identify the delay guards using a *top-down* approach, i.e., by initially setting them to be as general (large) as possible and specializing them (making them smaller) if necessary. The states and transitions are still identified using the conventional state-merging approach.

In our timed APTA (see Fig. 6), two timed strings end in the same state if their *untimed* strings (the strings obtained by disregarding the time values) are identical. We set the initial values of the lower and upper bounds of all delay guards of the timed APTA to be the

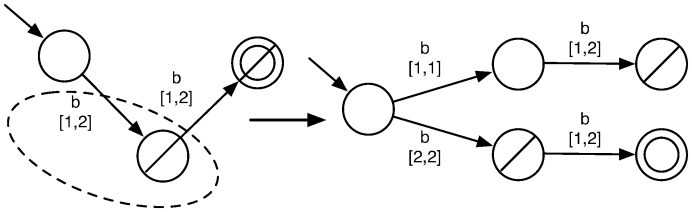

Fig. 7 A split of a part of the APTA from Fig. 6. *On the left*, the original DRTA is shown. The guard and target node that are to be split are surrounded by a *dashed ellipse*. *On the right*, the result of the split is shown. The split is called using time value $t = 1$

Algorithm 5 Splitting a transition: split

Require: an augmented DRTA $(A = \langle Q, \Sigma, \Delta, q_0, F\rangle, R)$, a transition $\delta = \langle q, q', a, [n, n']\rangle$, a time value $t \in [n, n' - 1]$, and an input sample S
Ensure: δ is split at time t and A is changed accordingly
Remove δ from A
Remove the part of the APTA with q' as root from A
Add two new states q_1 and q_2 to A
Add a new transition $\delta_1 := \langle q, q_1, a, [n, t]\rangle$ to A
Add a new transition $\delta_2 := \langle q, q_2, a, [t + 1, n']\rangle$ to A
Set q_1 to be the state reached by (a, t) in $A_1 := \mathsf{tapta}(S^{\delta_1})$
Set q_2 to be the state reached by $(a, t + 1)$ in $A_2 := \mathsf{tapta}(S^{\delta_2})$

minimum $t_{\min}$ and maximum $t_{\max}$ time values from the input sample S, respectively. This timed APTA is identical to the conventional APTA constructed from the untimed versions of the timed strings from S. We show a timed version of the APTA construction in Algorithm 4.

Our timed APTA construction allows for the possibility of *inconsistent* states. These are created when the untimed versions of a positive and a negative example are identical. For example, in Fig. 6, string bb is the untimed version of $(b, 2)(b, 1) \in S_+$ and $(b, 1)(b, 1) \in S_-$. Our algorithm can get rid of the aforementioned inconsistencies by *splitting* (specializing) a transition at some time value $t \in [t_{\min}, t_{\max} - 1]$.

Splitting a transition A *split* with time value t of a transition δ divides the part of the DRTA pointed to by transition δ into two parts. The first part is reached by the timed strings that fire δ with a delay value less or equal to t. The second part is reached by timed strings for which this value is greater than t. An example split is depicted in Fig. 7.

The exact result of a split depends on which timed strings from the input sample fire δ. Every such string can be written as $\tau(a, t')\tau'$, where τ is the prefix before firing transition δ, (a, t') is a pair containing the symbol a of δ and a time value t' that satisfies the delay guard $[n, n']$ of δ, and τ' is the suffix after firing δ. We call the timed string $\tau^\delta = (a, t')\tau'$ the *tail* of $\tau(a, t')\tau'$ for δ. Such a tail is said to be positive (negative) if $\tau\tau^\delta$ is positive (negative). We use S^δ to denote the subsample of all tails from S for δ, i.e., $S^\delta = (S^\delta_+ = \{\tau^\delta \mid \tau \in S_+\}, S^\delta_- = \{\tau^\delta \mid \tau \in S_-\})$. In a split the two divided parts of the DRTA are reconstructed using these tails. Because we use the red-blue framework, RTI can only split transitions that have blue states as their target. The splitting algorithm is shown in Algorithm 5.

Let δ be a transition to a blue node. Suppose S^δ contains two tails τ^δ_1 and τ^δ_2 that are equal if we disregard their time values. Initially, these two tails end in the same state due to the timed APTA construction. After a split of δ, however, it is possible that S^{δ_1} contains

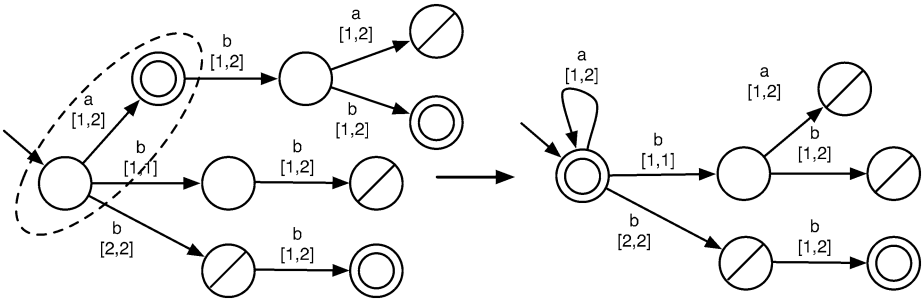

Fig. 8 A merge of a part of the APTA from Fig. 6 after the split from Fig. 7. *On the left*, the original DRTA is shown. The states that are to be merged are surrounded by a *dashed ellipse*. *On the right*, the result of the merge is shown

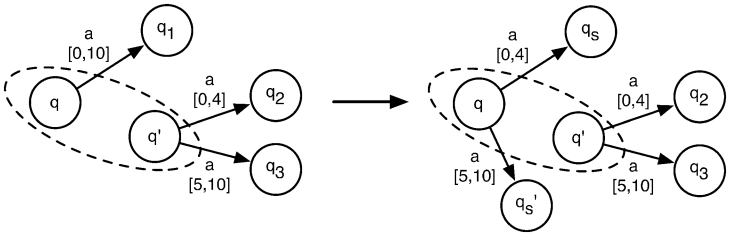

Fig. 9 When merging two states q and q', first the outgoing transitions are split in order to resolve non-deterministic choices that are due to differences in delay guards

τ_1^δ while S^{δ_2} contains τ_2^δ (or vice versa). In other words, τ_1^δ and τ_2^δ no longer end in the same state. In this way, a split can remove both consistent and inconsistent states from a DRTA. Our algorithm decides where to split a transition based on the amounts of removed consistent and inconsistent states. This is explained further in Sect. 5.

Merging states in a DRTA Besides the timed APTA construction and the split operation, our DRTA identification algorithm is still somewhat different from a conventional state-merging algorithm: the merge operation is modified in order to deal with the delay guards of a DRTA. Suppose that RTI wants to merge a blue state q and a red state q'. In the conventional state-merging algorithm, a new red state q'' will be created that has all the incoming and outgoing transitions of both q and q'. In the DRTA case, however, we cannot just use the outgoing transitions of both q and q' as the outgoing transitions for q'' because their guards can be partially overlapping. For example, suppose that a DRTA A contains the following transitions: $\langle q, q_1, a, [0, 10]\rangle$, $\langle q', q_2, a, [0, 4]\rangle$, and $\langle q', q_3, a, [5, 10]\rangle$, depicted in Fig. 9. If RTI merges q and q' into a new state q'', the transitions of q'' will be: $\langle q'', q_1, a, [0, 10]\rangle$, $\langle q'', q_2, a, [0, 4]\rangle$, and $\langle q'', q_3, a, [5, 10]\rangle$. Thus, there is a non-deterministic choice for a delay value in $[0, 4]$, with targets q_1 and q_2, and there is a different non-deterministic choice for a delay value in $[5, 10]$, with targets q_1 and q_3. In fact, because the guard of the transition to q_1 overlaps with both of the other transitions, a call to the determinization routine will merge all three of the states q_1, q_2, and q_3 into one single state. In other words, the determinization routine also merges the deterministic choice with targets q_2 and q_3. We solve this issue by splitting $\langle q, q_1, a, [0, 10]\rangle$ into $\langle q, q_s, a, [0, 4]\rangle$ and $\langle q, q_s', a, [5, 10]\rangle$ before merging q with q'. The guards of the outgoing transitions of q and q' are now identical

Algorithm 6 Merging two states: merge

Require: an augmented DRTA $(A = \langle Q, \Sigma, \Delta, q_0, F\rangle, R)$, two states q, q' from A such that q' is not red, and an input sample S
Ensure: q and q' are merged, A is updated accordingly, and *true* is returned, *false* is returned otherwise
  if it holds that $q \in F$ and $q' \in R$ or vice versa **then**
    return *false*
  end if
  Add a new state q'' to A that is neither accepting nor rejecting
  if it holds that $q \in F$ or $q' \in F$, **then** set $F = F \cup \{q''\}$
  if it holds that $q \in R$ or $q' \in R$, **then** set $R = R \cup \{q''\}$
  if there exist an outgoing transition $\delta' \in \Delta$ from q' **then**
    for all outgoing transitions $\delta = \langle q, q^*, a, [n, n']\rangle \in \Delta$ from q **do**
      if it holds that $n' \neq t_{\max}$ **then** call split$((A, R), \delta', n', S)$
    end for
  end if
  for all transitions $\delta = \langle q_1, q_2, a, [n, n']\rangle \in \Delta$ **do**
    if it holds that $q_1 = q$ or $q_1 = q'$ **then** set $q_1 := q''$
    if it holds that $q_2 = q$ or $q_2 = q'$ **then** set $q_2 := q''$
  end for
  while Δ contains two non-deterministic transitions $\langle q'', q_1, a, [n, n']\rangle$ and $\langle q'', q_2, a, [n, n']\rangle$ such that q_2 is not red **do**
    Boolean $b =$ merge$((A, R), q_1, q_2, S)$
    if b equals *false* **then**
      Undo the merge of q with q' and **return** *false*
    end if
  end while
  return *true*

and we can simply merge the two states. The determinization routine resolves the resulting non-determinism in the normal way.

Example 2 Figure 8 shows an example of a merge that includes a split operation before determinization. The second transition fired by $(a, 1)(b, 2)(b, 1)$ and $(a, 1)(b, 1)(a, 1)$ (labeled with b) is split first before starting the merge procedure. Notice that the accepting state gets merged with the bottom path (reached by the transition with guard $[2, 2]$) because it is reached by $(a, 1)(b, 2)(b, 1)$. The rejecting state gets merged with the top path because it is reached by $(a, 1)(b, 1)(a, 1)$.

In the red-blue framework, we can only merge a blue state with a red state. Consequently, the determinization routine only merges uncolored states with other states. Since we only split the outgoing transitions of red states, in any merge, all of the outgoing transitions of one of the two states is guaranteed to have the initial delay guard ($[t_{\min}, t_{\max}]$). Hence, we can always solve the determinization issue by simply splitting these outgoing transitions at the values of the delay guards of the other state. Algorithm 6 shows the merge and determinization routines that include these splits.

Algorithm 7 Checking permanent inconsistency: inconsistent

Require: A timed input sample S and an augmented DRTA $(A = \langle Q, \Sigma, \Delta, q_0, \rangle, R)$
Ensure: Returns *true* if (A, R) can still be made consistent with S
  for all red states q of Q **do**
    if it holds that $q \in F$ and $q \in R$ **then return** *false*
    for all transitions $\langle q, q', a, [n, n'] \rangle \in \Delta$ such that q' is not red **do**
      Let $S^\delta = (S^\delta_+, S^\delta_-)$ be the set of tails of S for δ
      if $S^\delta_+ \cap S^\delta_- \neq \emptyset$ **then return** *false*
    end for
  end for
  return *true*

4.3 The RTI algorithm for identifying DRTAs

In the previous section, we constructed routines for timed state-merging and transition-splitting. These routines can be inserted into the EDSM algorithm within the red-blue framework in order to construct an algorithm for identifying DRTAs. We call this algorithm RTI, which stands for Real-Time Identification. Algorithm 8 shows this algorithm. This algorithm tries all possible merges, splits, and colorings, in every iteration. It computes an evidence value (described in Sect. 5) for every possible action. The algorithm then performs the action that scores highest, but only if the result is not permanently inconsistent (Algorithm 7). An augmented DRTA is *permanently inconsistent* if either an inconsistency occurs in the red states, or if there exist an identical pair of a positive and a negative tail in the transitions to blue nodes. These tails can never be pulled apart by any subsequent split operation. If some actions score equally, the algorithm gives preference to first a merge, then a split, and finally a color operation.

It might be the case (and in fact it often is the case) that the evidence value gives the same score to a possible range of splits between two time values $t_1 < t_2$. We deal with such a situation by setting t equal to t_1. The motivation behind this is that if we should actually split it at some other time value $t < t' < t_2$, then the set of time values $[t + 1, t_2]$ is as large as possible. Hence, if at some later iteration of RTI there is some additional information (by performing merges and creating loops), then we can still identify the correct split later using as much information as possible. Setting t to $\lfloor \frac{t_1+t_2}{2} + 0.5 \rfloor$ might seem to make more sense, but this minimizes the additional information we can get in order to still identify the correct split if this split is incorrect. Note that incorrect splits can still be resolved by correct merges in subsequent iterations of RTI.

4.4 Properties of RTI

We now prove the following important properties of this algorithm: it is efficient, correct (sound), and complete. In addition, we show that under an appropriate evidence value, polynomial characteristic sets exist, and thus that RTI converges efficiently to the correct DRTA.

Proposition 1 RTI *is time efficient, i.e., it requires runtime polynomial in the size of the input sample* S.

Proof RTI starts with the construction of a timed APTA A. This construction (and the resulting APTA) is clearly polynomial in the size of the input. Then, in every iteration of RTI, either a transition is split, two states are merged, or a state is colored red in A. The split

Algorithm 8 Identifying DRTAs: RTI

Require: A timed input sample S, with alphabet Σ, and minimum and maximum time values $t_{\min}$ and $t_{\max}$
Ensure: A is a small DRTA that is consistent with S

```
Set (A, R) = (⟨Q, Σ, Δ, q_0, F⟩, R) = tapta(S, Σ, t_min, t_max)
Color the start state q_0 of A red
while A contains non-red states do
  for all transitions ⟨q_r, q', a, [n, n']⟩ ∈ Δ such that q_r is red and q' is not do
    Color q' blue
  end for
  for all transitions δ = ⟨q, q_b, a, [n, n']⟩ ∈ Δ such that q_b is blue do
    for all red states q_r in Q do
      Call merge((A, R), q_r, q_b, S)
      if inconsistent((A, R)) is false then
        Calculate the evidence value v_m
      end if
      Undo the merge of q_r and q_b
    end for
    Let S^δ be the set of tails for δ
    for all tails τ^δ = (a, t)τ' ∈ S^δ do
      Call split((A, R), δ, t, S)
      Calculate the evidence value v_s
      Undo the split of δ
    end for
    Color q_b red
    if inconsistent((A, R)) is false then
      Calculate the evidence value v_c
    end if
    Color q_b blue
  end for
  if A merge has the highest evidence value v_m then redo the merge
  else if A split has the highest evidence v_s then redo the split
  else redo the coloring with the highest evidence value v_c
end while
return A
```

operation requires three sets of timed strings S^{δ}, S^{δ_1}, and S^{δ_2}, which can all be constructed (or maintained in an efficient way using binary search trees) in polynomial time by running A on the input sample. The split operation uses two calls of the polynomial-time APTA construction in order to compute its result. The merge operation (including determinization) sometimes calls the polynomial split operation. Since every state created by these splits has to be reached by at least one timed string, the amount of extra states created is bounded by the size of the input sample. Thus, the merge operation combines a polynomial amount of states. Merging two states is clearly polynomial. Therefore, the merge operation is polynomial in the size of the input sample. Coloring a state only requires $O(1)$ time. This proves that each individual operation requires no more than polynomial time. What remains is to show that a single iteration of RTI requires no more than polynomial time and that the algorithm ends after a polynomial amount of iterations.

RTI only performs splits that separate the paths of two timed strings from the input sample. Hence, the amount of possible splits is polynomial in the size of the input sample, i.e.,

this amount is bounded by $|S| \times |S|$, where $|S| = \sum_{\tau \in S} |\tau|$. The amount of possible merges is bounded by the amount of possible pairs of states of the APTA, i.e., this amount is also bounded by $|S| \times |S|$. The number of times RTI colors a state red is bounded by the amount of possible states, i.e., it is bounded by $|S|$. Thus, in a single iteration, RTI can in the worst case try $2|S| \times |S| + |S|$ possible polynomial operations before deciding which action to take.

Once a state is colored red, a pair of states has been merged, or two timed strings have been split, the same action cannot occur again. Since one of these actions is performed in every iteration, this bounds the amount of possible iterations of RTI by $2|S| \times |S| + |S|$. Hence, the proposition follows. □

Proposition 2 RTI *is correct* (*sound*), *i.e.*, *given any input sample* $S = (S_+, S_-)$, *it returns a DRTA* A *that is consistent with* S (*such that* $S_+ \in L(A)$ *and* $S_- \in L(A^C)$).

Proof By definition, S is such that $S_+ \cap S_- = \emptyset$. Thus, although initially it is possible that A is inconsistent with S, it is not permanently inconsistent. RTI can make A consistent with S by splitting transitions at the appropriate time values. In addition, a color or merge operation is only performed if A is not permanently inconsistent afterwards. Hence, during the execution of RTI, A is never permanently inconsistent. More specifically, there exists no red state q_r such that $q_r \in F$ and $q_r \in R$. The timed APTA construction ensures that every timed string $\tau \in S_+$ ends in a state $q \in F$, and that every timed string $\tau \in S_-$ ends in a state $q \in R$.

RTI terminates only if all states are colored red. Hence, when it terminates, every timed string $\tau \in S_+$ ends in a red state $q_r \in F$, every timed string $\tau \in S_-$ ends in a red state $q_r \in R$, and there exists no red state q_r such that $q_r \in F$ and $q_r \in R$. In other words, when it terminates, A is consistent with S. Because the algorithm terminates after a polynomial number of iterations (Proposition 1), the proposition follows. □

We have just shown that RTI is sound and efficient. Thus, after an amount of time bounded by a polynomial in the size of the input sample, it will return a (small) DRTA that accepts all the positive timed strings and none of the negative times strings from the input sample. We now show that under the right evidence value RTI is also complete, and that it converges efficiently (in the limit from polynomial time and data). We prove this by showing the existence of polynomial characteristic sets (de la Higuera 1997):

Definition 3 (Polynomial characteristic set) A *polynomial characteristic set* S of a target language L_t for a learning algorithm $\mathcal{A}$ is an input sample $\{S_+ \subseteq L_t, S_- \subseteq L_t^c\}$ such that:

- the size of S is bounded by a polynomial in the size of the smallest model (automaton) A_t with $L(A_t) = L_t$;
- given S as input, algorithm $\mathcal{A}$ returns a model A such that $L(A) = L_t$;
- and given any input sample that contains S as input, $\mathcal{A}$ still returns A.

Intuitively, the existence of such sets for RTI shows that RTI requires only a polynomial amount of examples in order to return any target language L_t. Since this shows that RTI can return a DRTA for any language, completeness follows as a corollary. The result of RTI is not only determined by the input sample, but also by its evidence value. Therefore, we can only show these properties under a given evidence value. We use a very simple value that defines a strict ordering of the possible merges and splits. This ordering is similar to the one we used in previous work for proving the convergence of the ID_1DTA algorithm for learning one-clock DTAs (Verwer et al. 2011):

Definition 4 (Shortlex blue state) The *shortlex blue state* q_b in an augmented DRTA is the blue state reachable by the first timed string in shortlex order (that is $(a,t_1)(a,t_2)$ before $(a,t_3)(b,t_4)$, and (b,t_1) before $(a,t_1)(a,t_2)$). Ties are broken by the sums of their time values (that is $(a,t_1)(a,t_2)$ before $(a,t_3)(a,t_4)$ if $t_1+t_2<t_3+t_4$).

Definition 5 (ID_1DTA evidence) Let q_b be the shortlex blue state. Assign the value:

- 3 to any consistent merge with q_b,
- 2 to a consistent coloring of q_b, and
- 1 to the split of the transition to q_b with the unique time value t such that:
  - the resulting new shortlex blue state can be colored red (consistently), and
  - the shortlex blue state resulting from a split with time value $t+1$ cannot.

Assign 0 to all other operations.

When RTI uses this evidence value, it will first try to merge blue states, then color them if possible, and finally perform splits only if they are necessary. We now show that this strategy makes RTI converge efficiently to any target language L_t, i.e., requiring only a polynomial amount of data until it finds a DRTA A such that $L(A)=L_t$. We use the shortlex ordering for simplifying this analysis (using any other fixed ordering will have the same effect). Using evidence values based on data (Sect. 5) often results in more swift convergence in practice. Unfortunately, proving this convergence is very difficult. Even for DFAs, this type of proof only exists for fixed order algorithms such as RPNI (Oncina and Garcia 1992). Without a fixed order, much more complex convergence proofs based on statistics are needed, see e.g. Clark and Thollard (2004) for probabilistic DFAs.

We now continue with our proof of efficient convergence for RTI with ID_1DTA evidence. Our proof consists of two parts: an invariant showing the existence of examples that force RTI to perform correct merges, colorings, and splits (Proposition 3), which is then used to show that only a polynomial number of such examples are required to make sure that RTI identifies L_t (Lemma 1). Combined this shows the existence of polynomial characteristic sets (Theorem 2).

Proposition 3 *Given a target DRTA language L_t, there exists an input sample $S=\{S_+,S_-\}$ (with $S_+\subseteq L_t$ and $S_-\subseteq L_t^c$) for* RTI *with* ID_1DTA *evidence such that the following invariant holds*: *all the red states and transitions between them are part of a DRTA A_t such that $L(A_t)=L_t$.*

Proof For the initial case, only the start state q_0 of the timed APTA is colored red. The proposition above hold here, since any DRTA that accepts L_t also contains a start state. For the arbitrary case (induction step), assume that at the start of the current iteration there exists a DRTA A_t with $L(A_t)=L_t$ that contains all the red states as well as all the transitions between them. We distinguish three cases:

Case 1. RTI merges q_b with a red state q_r;
Case 2. RTI colors q_b red;
Case 3. RTI splits the transition δ to q_b.

In case 1, RTI identifies the target of a new transition $\delta=\langle q,q_r,a,[n,n']\rangle$. To arrive at a contradiction, suppose this transition is not included in A_t, however by the invariant both q and q_r are included. Without loss of generality, we assume that δ does not target q_r with a reason, i.e., adding δ to A_t and removing/modifying overlapping transitions results in a

DRTA A'_t such that $L(A'_t) \neq L_t$. If there is no reason, the invariant holds for A'_t instead of A_t, and A'_t is not larger than A_t. Under this assumption, there exist (at least) three timed strings τ, τ_r, and τ_d such that: τ ends in q, τ_r ends in q_r, and $\tau(a,n)\tau_d \in L_t$ but $\tau_r\tau_d \notin L_t$ or vice versa. If included in the input sample, $\tau(a,n)\tau_d$ and $\tau_r\tau_d$ will create a permanent inconsistency when RTI tries to merge q_b with q_r: after the merge, $\tau(a,t)$ and τ_r both end in q_r and the remaining τ_d has to be both rejected and accepted. Thus, by adding timed strings to the input sample S, we can force RTI to perform only merges that identify transitions δ that are included in A_t. Furthermore, if δ is included in A_t, such timed strings do not exist since $L(A_t) = L_t$.

In case 2, RTI identifies a new transition $\delta = \langle q, q_b, a, [n, n'] \rangle$ to a new red state q_b. Furthermore, due to the ID_1DTA evidence value, every single merge with existing red states resulted in a permanent inconsistency. Thus for all existing red states q_r, $\langle q, q_r, a, [n, n'] \rangle$ is not included in A_t. Suppose that δ is also not included in A_t. Since δ is the only transition that targets q_b, and since q_b cannot be merged with any existing red state, A_t does contain a transition from q to q_b with symbol a. Hence, A_t does not include such a transition with delay guard $[n, n']$. Instead, A_t includes such a transition with delay guard $[n_t, n'_t]$ such that $n_t > n$ and/or $n'_t < n'$. Therefore, without loss of generality, there exist two timed strings τ and τ_d such that: τ ends in q, and $\tau(a,n)\tau_d \in L_t$ but $\tau(a,n')\tau_d \notin L_t$ or vice versa. If included in the input sample S, these timed strings will create a permanent inconsistency when RTI tries to color q_b. Thus, we can also force RTI to perform only colorings that identify transitions δ that are included in A_t. Furthermore, if A_t does contain δ there clearly exists a timed string $\tau(a,t)$ that ensures that q_b is identified as a final state if and only if it is a final state in A_t. Hence, also the red final states are included as final states in A_t.

In case 3, RTI identifies the delay guard of a new transition $\delta = \langle q, q_b, a, [n, n'] \rangle$. Since this transition still targets a blue state q_b, the invariant clearly still holds after the split.

We conclude that if all red states and the transitions between them are included in A_t, then given the right input data, after an iteration of RTI, the red states and transitions between them are still included in A_t. By induction, all the red states and transitions between them remain included in A_t during the entire execution of RTI, and hence the proposition follows. □

What remains to show is that RTI will return a DRTA A_t with $L(A_t) = L_t$ after a polynomial number of iterations, and that during every iteration only a polynomial number of polynomially sized strings (a polynomial characteristic set) are required. Since RTI with ID_1DTA evidence performs the same steps as the ID_1DTA algorithm, the proof can immediately be derived from the fact that ID_1DTA identifies the larger class of 1-DTAs efficiently (Verwer et al. 2011). Here, we give the intuition of this proof, for details the reader is referred to the convergence proof of ID_1DTA.

Lemma 1 *There exists polynomial characteristic sets for* RTI *with* ID_1DTA *evidence.*

Proof (Sketch) Due to the fixed order over merges, colorings, and splits, we can generate a polynomial characteristic set for a target language L_t, with smallest target DRTA A_t, using the following method:

1. Start with an empty input sample S.
2. Run RTI with S as input one iteration at a time.
3. Whenever RTI performs a merge or coloring that identifies a transition not in A_t, add timed strings to S that create a permanent inconsistency after this operation (see Proposition 3), and restart RTI.

4. Otherwise, whenever RTI colors a blue state red, add a timed string that ends in the blue state to S, and restart RTI.
5. If RTI finishes without identifying such a transition, it returns A_t and S contains the characteristic set.

S contains a polynomial characteristic set because RTI follows a fixed order over merges, colorings, and splits. Because of this order, it will iterate through the exact same steps every time it is restarted. Only the last merge, coloring, or split operation will be different due to the examples added to S. In the end, every merge, coloring, and split operation identifies a transition, state, or/and delay guard in A_t. Since there cannot exist examples that create permanent inconsistencies when RTI performs merges or colorings resulting in transitions in A_t, this fixed (data independent) order ensures that adding extra examples to S does not influence the result of RTI (Definition 3 property 3).

By choosing examples on the boundaries of delay guards in A_t (see Proposition 3), the property of splits with evidence 1 in the ID_1DTA evidence value ensures that RTI identifies exactly the delay guards in A_t. Every other possible split receives an evidence of 0. The permanent inconsistencies ensure (without loss of generality, see Proposition 3) that the transition targets are also contained in A_t. Thus every transition of A_t will be identified by RTI, and since $\mathcal{A}_t$ is finite, this ensures that RTI will return A_t (Definition 3 property 2).

Since RTI cannot identify extra delay guards, the number of splits performed by RTI will be bounded by the number of transitions in A_t.

In every iteration, before finding the operation that identifies a transition in A_t, RTI will try a number of merges and colorings bounded by the number of states in A_t. Since the number of transitions RTI identifies is bounded by the number of transitions in A_t, the above method requires a polynomial number of restarts before finishing. Every time it restarts it adds at most two timed strings of polynomial length (see Verwer et al. 2011) to S, and thus the size of S is bounded by a polynomial in the size of A_t (Definition 3 property 1). This proves the lemma. □

Corollary 1 RTI *is complete, i.e., for every DRTA language L there exists a sample S such that* RTI *returns a DRTA A with $L(A) = L$.*

Proof If polynomial characteristic sets (Definition 3) exist for any DRTA target language L_t (Lemma 1), RTI is able to return a DRTA A such that $L(A) = L_t$. □

Theorem 2 RTI *with* ID_1DTA *evidence converges efficiently (from polynomial time and data) to any DRTA language in the limit.*

Proof By the fact that RTI is efficient (Proposition 1) and the existence of polynomial characteristics sets for RTI (Lemma 1). Together these form the definition of identification in the limit from polynomial time and data (see, e.g., de la Higuera 1997), i.e., efficient convergence. □

In conclusion, RTI is a timed version of EDSM that runs in polynomial-time, is a correct and complete algorithm, and converges efficiently to the correct DRTA in the limit using ID_1DTA evidence. Such a fixed order value, however, does not perform very good in practice since it does not use any of the statistical information available in the data set. Therefore, we base the heuristics for RTI on the value used by EDSM. In the next section, we provide a few of these values.

5 Heuristics for RTI

Our algorithm uses an evidence (score) value (measure) in order to determine which action to take. The action that results in the highest evidence value, i.e., the one that agrees most with the input sample, is selected to be performed. Thus, the evidence value can be thought of as a *heuristic* that guides RTI. RTI stops when it has reached a local optimum, i.e., when it cannot perform any actions and can therefore be thought of as a greedy procedure. In this section, we describe the four heuristics for this greedy procedure that we use in our experiments. In addition, we explain a simple search variant of RTI. This search procedure uses the size of the DRTA resulting from an entire run of RTI as an evidence value. We compared the performance of this search variant in our experiments in order to give some insight into the effects of searching for a smaller solution.

5.1 Four evidence values

Since RTI is based on the EDSM algorithm for the identification of DFAs, we also based all of our heuristics on the EDSM evidence value. The intuition behind the EDSM evidence value is that the labels of the states that are combined during the determinization procedure of a merge can be seen as statistical tests for testing whether the merge is good or bad. A good merge is one where the states are instances of the same state in the target DFA, in a bad merge they are instances of different states. In the case of a good merge, none of the labels of the states that are merged by the determinization procedure can result in an inconsistency. In the case of a bad merge, some of these labels can result in an inconsistency. Thus, the greater the number of merged labeled states, the more confident we are that the merge is a good merge.

In addition to the normal non-timed (EDSM) evidence, RTI has access to information in the form of time values. Naturally, we would like RTI to make use of this timed information. Moreover, we would like to show that using this additional information leads to improved performance.

We created four heuristics that make use of the EDSM evidence value and the additional time information in different ways. We now first explain the two simple ones that do not make use of time information, then one that does make use of time information by giving more power to tails that are closer together, and finally one that tries to penalize the evidence value based on the amount of splits necessary in order to remove all inconsistencies.

Pure EDSM In order to give more insight into whether it is beneficial to make use of the time information in the input sample, we included the exact score value used by EDSM in our experiments. The evidence value used by EDSM is defined as:

$$\mathsf{pure} = \#\mathsf{merge}(pos, pos) + \#\mathsf{merge}(neg, neg)$$

where $\#\mathsf{merge}(pos, pos)$ (or $\#\mathsf{merge}(neg, neg)$) is the number of merges of pairs of only positive (or negative) states performed so far (including by the determinization procedure). We compute this value before and after a merge or split operation, and use the difference as a heuristic value. Since a split can only make previous merges undone, a split can only obtain negative values, and thus this heuristic performs splits only if no merge or coloring is possible.

Consistent EDSM A big difference between EDSM and our DRTA identification algorithm is that in our case, we allow for the possibility of inconsistent states. The split operation can be used to remove such inconsistencies. Intuitively, these inconsistencies should of course have some negative influence on the evidence value. We included a variant of EDSM that uses the inconsistencies in a very simple way: it simply subtracts the number of inconsistent merges from the number of consistent ones. It is defined as:

$$\mathsf{consistent} = \#\mathsf{merge}(pos, pos) + \#\mathsf{merge}(neg, neg) - \#\mathsf{merge}(pos, neg)$$

When using this evidence value it is possible that a color operation gets the highest score. This occurs when all possible splits and merges score negatively (when they add (remove) more (less) inconsistent merges than they add (remove) consistent merges). This is different from conventional EDSM within the red-blue framework where a color operation is only applied when no merge is possible. Our algorithm simply picks the highest scoring operation, including color operations. If some operations score equally, we give preference to first merge, then split, and finally color operations.

Impact EDSM The first timed value we use is based on the idea that tails that lie far away from each other are more likely to be pulled apart by a future split operation than tails that lie close to each other. For example, suppose that RTI merges two states in a DRTA, each containing just one tail (each state is reached by just one timed string). Let these tails starting from these two states be something like: $(a, 1)(b, 3)(c, 5)$ and $(a, 2)(b, 3)(c, 4)$. These tails lie close to each other in time and should intuitively get a higher impact on the score than say: $(a, 1)(b, 3)(c, 5)$ and $(a, 5)(b, 6)(c, 7)$.

We calculate this impact value using the distance in time between every pair of tails τ and τ' that are identical if we disregard their time values, i.e., if $\mathsf{untime}(\tau) = \mathsf{untime}(\tau')$. These two tails currently end in the same state in the augmented DRTA $(A = \langle Q, \Sigma, \Delta, q_0, F\rangle, R)$. We define the impact of τ and τ' as the probability that τ and τ' still end in the same state if we were to choose a split point uniformly at random in every non-red transition. Since RTI cannot split transitions between two red states, these transitions are excluded from this definition. The impact of two tails is calculated using Algorithm 9. The evidence value we use is computed as the sum over all timed strings from S of the maximum impact with any other consistent timed string minus the maximum impact with any other inconsistent timed string. All pairs of timed strings that end in a red state get an impact of 1.0 because they can never be separated by any split. Let Q_r denote the red states of Q, and let Δ_b denote the transitions to blue states of Δ. S^δ denotes the sample of tails (suffixes) of timed strings from S that fire δ. Using these sets, the evidence value is:

$$\begin{aligned}\mathsf{impact} = &\sum_{q\in Q_r} \mathsf{pure}(q) - \sum_{\delta\in\Delta_b} \max\left\{\mathsf{impact}(\tau,\tau') \mid \tau \in S^\delta_+ \text{ and } \tau' \in S^\delta_-\right\}\\ &+ \sum_{\delta\in\Delta_b} \max\left\{\mathsf{impact}(\tau,\tau') \mid \tau \neq \tau', \tau \in S^\delta_+ \text{ and } \tau' \in S^\delta_+\right\}\\ &+ \sum_{\delta\in\Delta_b} \max\left\{\mathsf{impact}(\tau,\tau') \mid \tau \neq \tau', \tau \in S^\delta_- \text{ and } \tau' \in S^\delta_-\right\}\end{aligned}$$

where $\mathsf{pure}(q)$ is the pure evidence value for state q, i.e., the number of timed strings from S that end in q minus 1.

Algorithm 9 The probability of not separating two timed strings: IMPACT

Require: Two tails τ and τ' for a transition to a blue state δ that have equal untimed versions, and the minimum and maximum time value $t_{\min}$ and $t_{\max}$
Ensure: Returns the probability that τ and τ' end in the same state if we split all non-red transitions uniformly at random
Set probability $p := 1.0$
for all integers $1 \leq i \leq |\tau|$ **do**
Let t_1 and t_2 be the ith time values in τ and τ' respectively
Set $p := p \times (1.0 - \frac{|t_1 - t_2|}{t_{\max} - t_{\min}})$
end for
return p

Algorithm 10 Making a state consistent with splits: splits

Require: A state non-red q of a colored DRTA A and a timed input sample S
Ensure: Returns an approximation of the minimum amount of times we need to split a transition in order to make q consistent
Let δ be transition to the root of the APTA that contains q
Let $S^q = (S^q_+, S^q_-)$ denote the subsample of S^δ of tails for δ that end in q
for all integers $1 \leq i \leq n$, where n is the length of the tails of S^q **do**
Let T_+ denote the set of ith time values in tails from S^q_+
Let T_- denote the set of ith time values in tails from S^q_-
if $T_+ \cap T_- = \emptyset$ **then**
Set $s := 0$
for all time values $t \in T_+$ **do**
If $\min(\{t' \in T_- \mid t' > t\}) < \min(\{t' \in T_+ \mid t' > t\})$ **then** set $s := s + 1$
end for
for all time values $t \in T_-$ **do**
If $\min(\{t' \in T_+ \mid t' > t\}) < \min(\{t' \in T_- \mid t' > t\})$ **then** set $s := s + 1$
end for
end if
end for
return the minimum value for s

Splits EDSM Like our second evidence value, our last evidence value includes inconsistencies in an EDSM-like score. However, it does so in a way that is a bit more sophisticated than simply subtracting the amount of inconsistencies. Instead, it removes these inconsistencies from the timed APTA by splitting transitions. Afterwards it uses the standard EDSM evidence value on the consistent APTA. The intuition is that an APTA is less likely to be correct (perform well) if it is more difficult to remove inconsistencies. In this case, more splits will be required, thus more merges will be undone, and the resulting EDSM score will be smaller.

The size of the resulting consistent APTA, and hence the resulting EDSM score, is determined by the splits performed by the algorithm that computes the evidence value. Unfortunately, the problem of finding splits that minimize the size of the resulting APTA is difficult. In fact, the hardness proof for determining the correct delay guards (the proof of Theorem 1) can easily be adapted to show that this problem is NP-hard. We therefore use a simple approximation algorithm to determine the splits (Algorithm 10). This algorithm computes the minimum amount of splits required to make one state of the APTA consistent. We use this

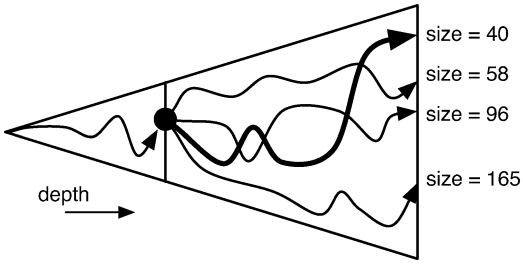

Fig. 10 The search tree of our simple search process depicted as a triangle, the search depth increases *from left to right*. From the current node of the search tree, the search procedure computes a complete execution of RTI for every possible action, and chooses the action that results in the smallest DRTA (size = 40)

algorithm to compute the following recursion for every state q:

$$\#\mathsf{splits}(q, S, A) = \begin{cases} \max(\mathsf{splits}(q, S, A),\ \#\mathsf{splits}(q', S, A)) & \text{if } q \text{ is not red} \\ 0 & \text{otherwise} \end{cases}$$

where q' is the source state of the transition to q. Intuitively, this value reflects the amount of states that a state q' will be split into if we were to remove all inconsistencies from A. If the parent state q of q' requires more splits than q', we assume that q' will also be split into the same amount of states. Since red states cannot be split, their evidence value is 0. In order to obtain an estimate on the number of remaining consistent merges we simply subtract $\mathsf{splits}(q, S, A)$ from the original amount of consistent merges of states (the pure evidence value), with 0 as a minimum. For the complete augmented DRTA ($A = \langle Q, \Sigma, \Delta, q_0, F\rangle, R$), with input sample S_+, the evidence value then becomes:

$$\mathsf{splits} = \sum_{q \in Q} \max\big(\mathsf{pure}(q) - \#\mathsf{splits}(q, S, A), 0\big)$$

where $\mathsf{pure}(q)$ is the pure evidence value for state q. This split evidence value does not compute the actual EDSM evidence that remains if we were to split our DRTA in an optimal way, but it tries to approximate this value from above.

5.2 A simple search procedure

In addition to these four heuristics, we tested a simple search process that we wrapped around RTI in order to test how much we can increase the performance by searching. Such an approach has been successfully applied to the original EDSM algorithm (Bugalho and Oliveira 2005). Our search process uses an evidence values in an indirect way (see Fig. 10):

- For every possible action (split/merge), the search procedure first computes a complete (greedy) execution of RTI with an evidence value. The result is a DRTA A.
- The search process then chooses the action that results in the smallest DRTA, i.e., such that the number of transitions in A is smallest.

The intuition behind this process is that actions taken by RTI that lead to a small DRTA are more likely to be correct. More specifically, although an evidence value m assigns a small value to a specific action a, if a is in fact a good (correct) action, then the result of performing a and then using m to determine the subsequent actions can still lead to a better

result than an action with a high value. In other words, the action that scores highest using m is not always the action that leads to the best result after a complete greedy run using m. Like our original algorithm, once the algorithm has chosen an action, it cannot be changed by subsequent iterations. The main benefits of this simple search procedure compared to a search procedure that uses an evidence value directly are:

- The procedure is an anytime algorithm. We can remember the smallest resulting DRTA and return it if the process is interrupted.
- The procedure produces results that are at least as good as the non-search (greedy) algorithm.
- The procedure requires only polynomial time in the size of the input sample: There are at most a polynomial amount of possible actions and the non-search algorithm is a polynomial time algorithm.
- The procedure ends without having to search through all possible DRTAs smaller than the result, i.e., it is incomplete but efficient.

Although our search procedure is a polynomial time algorithm, it takes a lot more time than the execution of our original algorithm since it can call this original algorithm tens of thousands of times as a subroutine.

6 Experiments

In this section, we experimentally compare each of the different evidence values (including the search procedure) for RTI described in the previous section. In addition, we compare all of these implementations of RTI to a sampling approach that first samples the timed data and then runs EDSM to obtain a DFA representation of a DRTA. We perform the experiments on a large set of artificial data sets generated from a randomly generated DRTA. In order to give insight into which method performs best in which setting, we generate these DRTAs with different number of states, transitions, delay guards, time values, etc.

This section is structured as follows. We first describe the sampling approach in Sect. 6.1. Then, we explain the experimental setup and provide the algorithms we used to produce the artificial data sets in Sect. 6.2. We list our expectations with respect to these experiments in Sect. 6.3. Finally, we show, describe, and explain our results in Sect. 6.4.

6.1 Sampled finite automata

As mentioned in the introduction, it is possible to sample timed data into an equivalent untimed format. Suppose we have a timed string $\tau = (a_1, t_1) \cdots (a_n, t_n)$. The equivalent untimed format for this timed string is a string $s_1 a_1 \cdots s_n a_n$, where $s_i = \circlearrowleft\circlearrowleft \cdots \circlearrowleft$ is a string consisting of special time tick symbols $\circlearrowleft$ of length t_i.

Similarly, there also exists an equivalent DFA representation for any DRTA language. In fact, there exists a DFA representation for any DTA language. Given a DTA, a DFA representation can be constructed using the region construction method (see Alur and Dill 1994). Essentially, this method creates additional states for every possible combination of a state and a time value. The same state with a higher time value can be reached by a series of transitions labeled with the time tick symbol $\circlearrowleft$. In Fig. 11, we give the DFA that results from the region construction when applied to the DRTA of Fig. 1. Formally, we can use the region construction to show that the language of any DRTAs can be recognized by a DFAs when the timed strings are sampled:

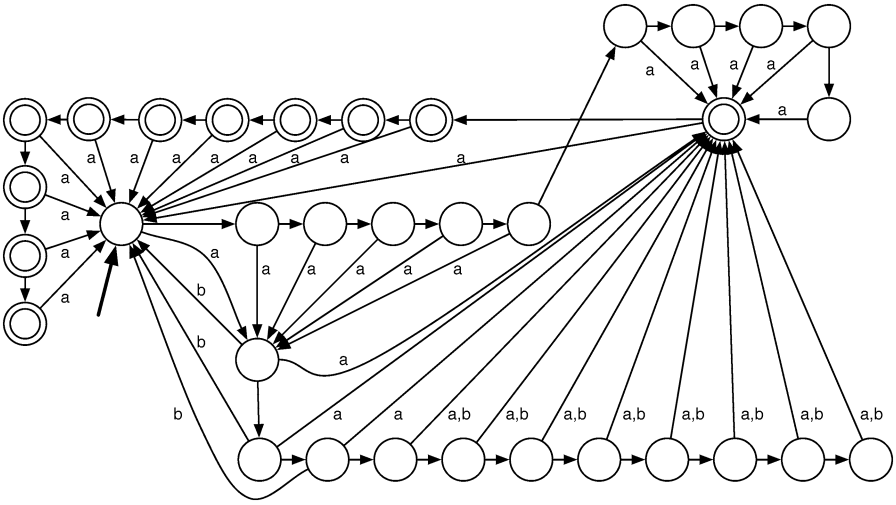

Fig. 11 A DFA equivalent to the harmonica DRTA of Fig. 1. The start state is indicated by the *thick arrow*. The *arrows with no label* are the time tick transitions

Lemma 2 *For any DRTA A there exists a DFA A' such that for every timed string τ it holds that $\tau \in A$ if and only if* $\text{SAMPLE}(\tau) \in L(A')$.

Proof Let $A = \langle Q, \Sigma, \Delta, q_0, F\rangle$ be a DRTA, and let $c \in \mathbb{N}$ be the largest constant that occurs in any delay guard of A. We define the following DFA $A' = \langle Q', \Sigma', \Delta', q_0', F'\rangle$, where:

- $Q' = \{\langle q, i\rangle \mid q \in Q, i \in \mathbb{N}, \text{ and } 0 \leq i \leq c+1\}$
- $\Sigma' = \Sigma \cup \{\circlearrowleft\}$
- $\Delta' = \{\langle\langle q, i\rangle, \langle q, i+i\rangle, \circlearrowleft\rangle \mid \langle q, i\rangle, \langle q, i+1\rangle \in Q'\} \cup \{\langle\langle q, i\rangle, \langle q', 0\rangle, a\rangle \mid \exists\langle q, q', a, [n, n']\rangle \in \Delta$ such that $i \in [n, n']\}$
- $q_0 = \langle q_0, 0\rangle$
- $F' = \{\langle q, 0\rangle \mid \langle q, 0\rangle \in Q'$ and $q \in F\}$.

We now claim that for every timed string τ it holds that $\tau \in L(A)$ if and only if $\text{SAMPLE}\tau \in L(A')$. We prove this by showing that the following invariant holds for any prefix τ_i of τ: if τ_i ends in q in A, then $\text{SAMPLE}(\tau_i)$ ends in $\langle q, 0\rangle$ in A'.

For the initial case, A starts in q_0 and A' starts in $\langle q_0, 0\rangle$. For the arbitrary case, let (a, t) be the ith symbol-time value pair of τ, i.e., $\tau_i = \tau_{i-1}(a, t)$. Suppose for the sake of induction that τ_{i-1} ends in q in A and that $\text{SAMPLE}(\tau_{i-1})$ ends in $\langle q, 0\rangle$ in A'. There has to exist a transition $\langle q, q', a, [n, n']\rangle \in \Delta$ such that $t \in [n, n']$. Thus, τ_i ends in q'. By construction of A', there also exists a regular transition $\langle\langle q, t\rangle, \langle q', 0\rangle, a\rangle$ in A'. In addition, there exists time tick transitions $\langle\langle q, i\rangle, \langle q, i+1\rangle, \circlearrowleft\rangle$ for all $0 \leq i \leq t$ in A'. The sampling transformation of (a, t) results in a string $s = \circlearrowleft \cdots \circlearrowleft a$ such that $|s| = t + 1$. This string s will fire t time tick transitions and one regular in the computation of A' over $\text{SAMPLE}(\tau)$. After firing the t time tick transitions, it will have reached the state $\langle q, t\rangle$. Once in this state, it will fire the regular transition $\langle\langle q, t\rangle, \langle q', 0\rangle, a\rangle$, thus ending in $\langle q', 0\rangle$. The invariant holds by induction.

Thus, if τ ends in a state q in A, then $\text{SAMPLE}(\tau)$ ends in a state $\langle q, 0\rangle$. By construction of F', if $\tau \in L(A)$ then it holds that $\text{SAMPLE}(\tau) \in L(A')$. This proves the lemma. □

Since region construction can always be used to construct a DFA that accepts exactly the same language as a DRTA, it is never really necessary to identify a DRTA. Instead we can identify a DFA that recognizes exactly the same language. The downside to trying to identify this DFA is that it is exponentially larger in the size of the DRTA. More specifically, since all constants of a DRTAs are written down in binary, the number of states is exponential in the size of these constants. If we want the DFA returned by a DFA identification algorithm to be correct, we naturally require that every state in this DFA is reached by at least one string from the input data. Hence, using a DFA identification algorithm in order to identify a DRTA language is inefficient since it requires an exponential amount of data in order to return the correct result. In other words, it requires exponential time and space in order to converge. In contrast, by Lemma 2, it is possible to converge to the correct DRTA using only a polynomial amount of data.

In theory, identifying a DRTA model for a DRTA language seems to be a better idea than identifying a DFA model for a sampled DRTA language. However, in practice one may argue that it makes no sense to sample the timed data using single time ticks. If the timed data has millisecond precision, it probably does not hurt to sample the timed data every 100 or even 1000 time ticks. Hence, in practice, the sampling transformation will usually contain an additional argument r that denotes the *sampling rate*. The result of sampling a timed string $\tau = (a_1, t_1)(a_2, t_2)(a_3, t_3) \cdots (a_n, t_n)$ with rate r is a string $s_1 a_1 s_2 a_2 s_3 a_3 \cdots s_n a_n$, where $s_i = \circlearrowleft\circlearrowleft \cdots \circlearrowleft$ is a string of length $\lfloor \frac{t_i}{r} + 0.5 \rfloor$. Using such a sampling rate, it is of course possible that the region construction no longer works. In other words, it is possible that there exists no equivalent DFA for a DRTA when the timed strings are sampled with rate r. However, the blowup in automaton size is not so great when this rate r is used (the blowup is exponential in $\frac{c}{r}$). In practice, it is not a big problem that the resulting DFA is not equivalent to the actual DRTA. We are usually satisfied when it only approximates (in terms of a high percentage of overlap in accepted timed strings) the actual DRTA.

In the next section, we describe how we compare the RTI algorithm with the approach of first sampling the timed data using some fixed frequency and then using a DFA learning algorithm.

6.2 Experimental setup

In order to test our DRTA identification algorithms, we generate random DRTAs with 4, 8, 16, and 32 states and alphabets of sizes 2, 4, and 8. To each of these DRTAs, we apply the split routine 4, 8, 16, and 32 times at random time values in order to create clock guards. The minimum time value of the clock guards is set to 0. The maximum time values of the clock guards is either 100 or 1000. Every state of the DRTA has a chance of 0.5 to be an accepting state. We disallow the case that all or none of the states were chosen to be accepting.

From these DRTAs, we randomly generate input samples consisting of either 1000 or 2000 timed strings. These input samples are used as data sets for our algorithms. We also generate test sets consisting of 50000 newly generated timed strings. Each of these timed strings has a probability of 0.1 to stop in each state it visits. Whether a string is positive or negative was determined by the state it ended in.

For every unique combination of the parameters of the DRTA generating algorithm, we generate 10 different DRTAs. This results in $4 \times 3 \times 4 \times 2 \times 2 \times 10 = 1920$ data and test sets. In order to compare RTI with a sampling approach, we sampled these data and test sets using a fixed sampling size of 10. We replace every symbol-time value pair (a, t) with an a symbol and $\frac{t}{10}$ special time increase symbols. Rounding is used to get rid of fractions. We use a sampling size of 10 for both the 100 and 1000 maximum values of the clock guards.

We run RTI on the constructed data sets, and used the test set to evaluate its performance. For the search process, we use the consistent EDSM value because it is simple to compute, and because it turned out to perform not worse than the timed evidence values. The algorithm we use for identifying a DFA from the sampled data is the red-blue algorithm, which we downloaded from the Abbadingo web-site.[1] In the following, we first present the expected results, then we provide and discuss the actual results of these algorithms.

6.3 Expectations

The main goal of the experimental setup described in the previous section, is to determine whether RTI performs better than the sampling approach. Because sampling results in a blow-up in both the amount of data and the size of the resulting automaton, we expect that RTI will perform better. In addition, the sampling transformation results in a small loss of information (otherwise there would be an even greater blow-up). However, since the sampling approach is a straightforward application of the current state of the art in DFA identification, we expect that this approach also performs reasonably well.

With respect to the four different evidence values, we expect that both the splits and impact EDSM values will perform best because they make use of the time values of timed strings. The pure and consistent values do not consider these values. Between the two of them, we expect the consistent value to perform better because it also uses the inconsistent merges as information. We do not know what to expect regarding the performance difference of the impact and splits values. Both values make sense, we leave it to the experiments to show which one is superior.

When increasing the amount of delay guards (splits) of the original DRTA that was used to produce the data sets, we expect the decrease in performance of the timed evidence values to be less than the non-timed evidence values. This makes sense because additional splits cause more differences in behavior between timed strings with the same symbols, but different time values. Similarly, when increasing the amount of states, we expect the non-timed evidence values to decrease more slowly than the timed ones, because the behavior is then not influenced by differences in time values. The same holds for additional symbols in the alphabet.

We expect that all methods perform better when more data is available, and that all perform worse when more time points are used. The search version of RTI should perform best, since this one runs a greedy version of RTI many times in order to decide between a single merge or split. We hope that, using the search procedure, we will be able to identify DRTAs that are sufficiently large enough for practical applications.

6.4 Results

We run RTI on all of the 1920 data sets using each of the different evidence values. In addition, we run the sampled and the search approaches on these data sets. We use two indicators for the performance of the DRTAs (or DFAs in case of the sampling method) resulting from these runs. The first is the size, measured in number of transitions of the DRTA (DFA), which we like to be as small as possible. The second is the percentage of correctly classified timed strings of the test set, which we like to be as large as possible. In Fig. 12, we show both of these performance indicators for every method over all data sets

[1] Abbadingo One: DFA Learning Competition: http://abbadingo.cs.unm.edu/.

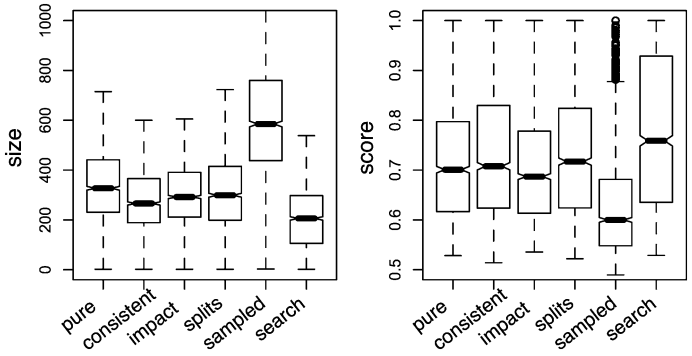

Fig. 12 Boxplots of the sizes and scores of every method over all data sets

as *boxplots*. The plots show the lower quartile, median, and upper quartile as a box. Around the median of each box, there is a small triangular cut in the box, called a *notch*. This notch depicts (roughly) the 95% confidence interval of the median. Thus, if the median of another boxplot is outside of this range, then the median of these two plots differ significantly.

From Fig. 12, it is clear that the sampling method performs much worse (in both performance measures) than RTI, independent of which of the evidence values is used. In addition, the search procedure does help to increase the performance of RTI (again in both measures). Furthermore, the consistent evidence value results in smaller DRTAs on average than any of the other values. We used a paired t-test for independence to test the significance of this difference in size between the consistent and splits evidence values. The result of this test is a p-value of less than $\times 10^{-16}$, thus the difference is significant. On average, the consistent evidence value finds DRTAs that contain 36 transitions less than the splits evidence value.

It seems that the score of the consistent value is smaller on average than the score of the splits value, but this score difference is not significant: a paired t-test applied to these score values results in a p-value of about 0.666. Surprisingly, the impact value performs the worst of the four evidence values. The score of the impact value is even (just) below the pure evidence value. Apparently, the distance in time between tails is a less successful heuristic than simply counting the number of correct merges. Since the impact value does make use of the time information in a sensible way, this is an unexpected result.

When looking at the boxplots in Fig. 12 a question that comes to mind is whether it is better to return smaller DRTAs. The results of the search method seem to indicate that searching for smaller solutions only performs a little worse in a few cases, but in much more cases it performs a lot better. More specifically, searching performs worse in 440 of all 1920 cases. The average decrease in score of these 440 cases is 0.01. However, in the 1480 remaining cases the average score increase is 0.05. To give more insight into the relation between score increase and size decrease, we created Fig. 13. Figure 13 shows the decrease in size when searching plotted against the score increase. Again, each individual dot in this plot represents a single data set. From this plot we conclude that finding smaller DRTAs helps a lot, but only if the size decrease is more than approximately 25 transitions.

In the remainder of this section, we show plots of the performance of every method on different settings of each individual parameter that was used to generate the data sets. This can help to give insight into which method should be used in which setting. It is especially interesting to find out with what settings the splits value outperforms the consistent value and the other way around. In addition, these results show how each of the parameters influences the difficulty of the identification problem.

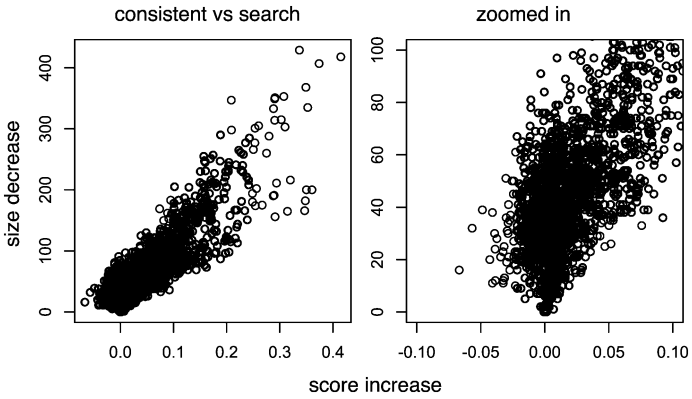

Fig. 13 The size decrease plotted against the score increase when searching. The *plot on the right* shows a zoomed-in version of the *left plot*

Different sized target DRTAs Figures 14, 15 and 16 show the performance of each of the individual methods on data sets for different sizes (alphabet, states, and splits) of original DRTAs that were used to generate the data set. We observe that the sampling method performs the worst in all cases. This is a very good result as it shows the value of the RTI algorithm. The search procedure performs best overall. It scores sufficiently good (about 80% accuracy) for DRTAs with up to 16 states, 16 splits, and alphabet of size 2, for DRTAs with up to 16 states, 8 splits, and alphabet of size 4, and for DRTAs up to 4 states, 8 splits, for an alphabet of size 8. Showing that RTI can be successfully applied in practice for problems of this size.

For the larger DRTAs, however, the search procedure does not significantly improve the performance of RTI. All methods perform bad on problems with a large number of states, and even worse on problems with a large number of splits. However, when a large amount is states is combined with a large alphabet, all methods perform better in the case of many splits than in the case of many states. This makes sense as each additional state creates an additional number of transitions to identify equal to the alphabet size.

With respect to the heuristic performances, we observe that on alphabets of size 2, the splits value outperforms the consistent value both on the resulting size and the score values. On alphabets of size 8, this is the other way around. This suggests that the splits value performs better on the smaller problems and the consistent value performs better on the larger problems. These results also show the difficulty of identifying the correct splits. Intuitively, when the number of splits is high, the evidence value that make use of time information should perform better than those that do not. This is exactly what we observe. With few splits, the scores of both the splits and the impact values score are lower than the score of the consistent value. With increasing amounts of splits, however, the difference in score becomes less. At 16 splits, the splits value already outperforms the consistent value. At 32 splits, even the impact value starts to outperform the consistent value. In fact, this is the only plot we have where the impact value outperforms any of the other values.

Different number of time values and data set sizes Figure 17 shows the performance of each of the individual methods on data sets while varying the amount of examples, and the amount of time values. From these plots we draw two important conclusions:

– More data leads to larger DRTAs, but the score of these DRTAs is significantly larger. This also holds for the sampling approach.

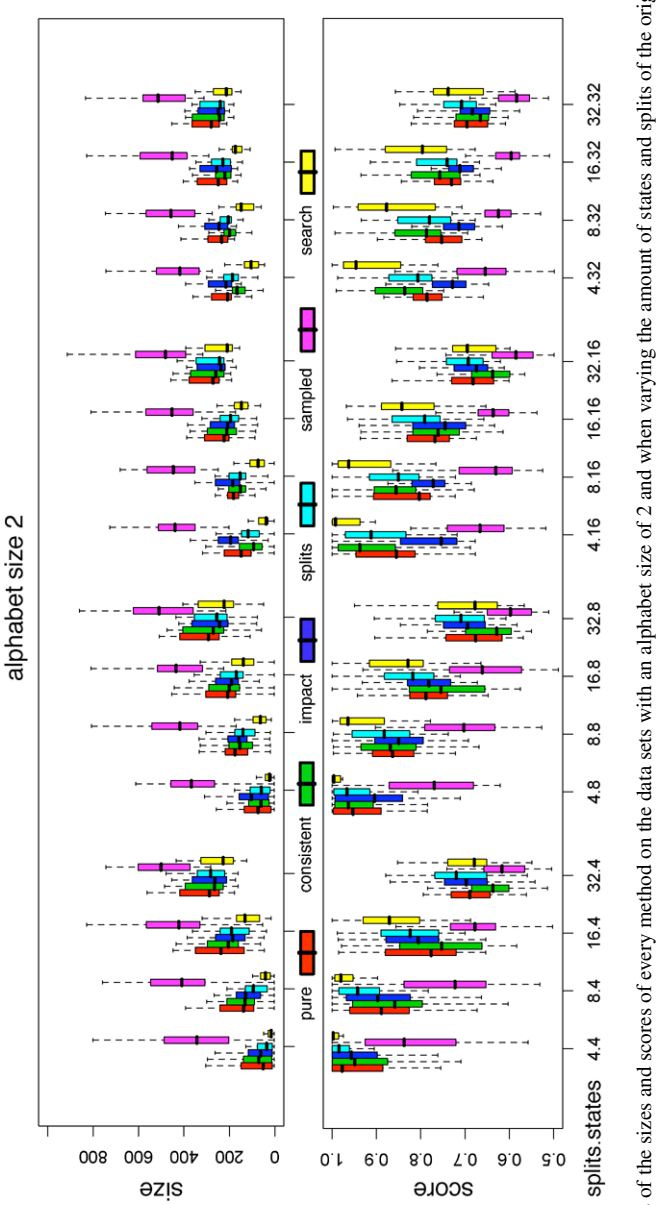

Fig. 14 Boxplots of the sizes and scores of every method on the data sets with an alphabet size of 2 and when varying the amount of states and splits of the original DRTA

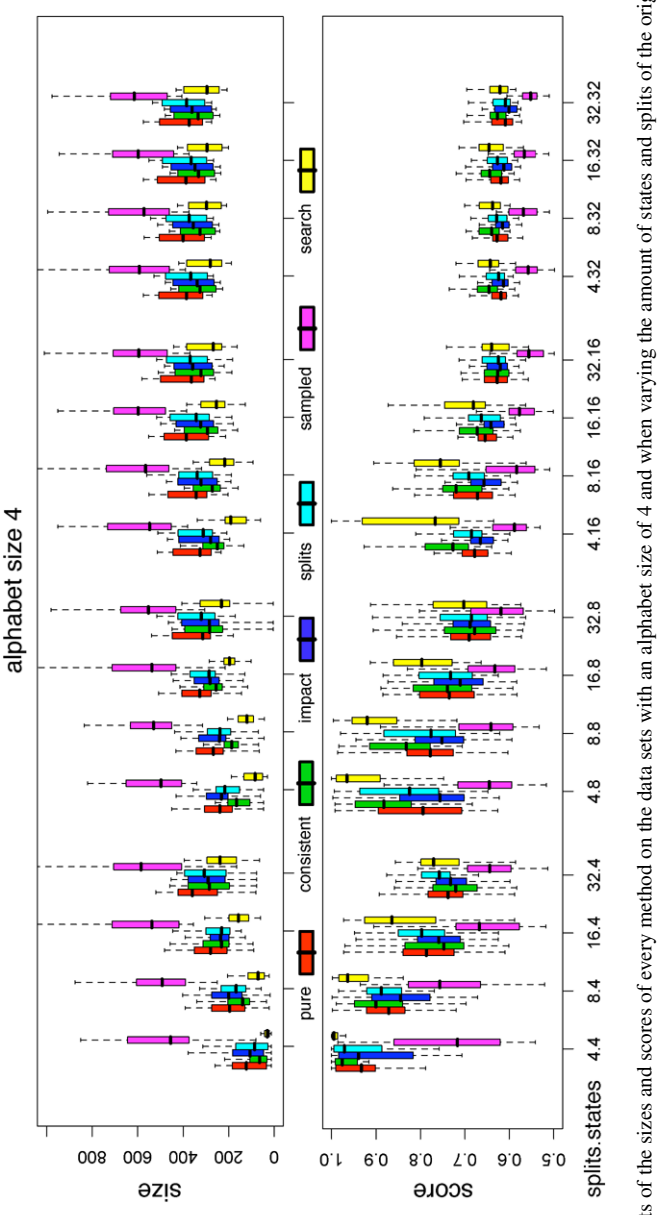

Fig. 15 Boxplots of the sizes and scores of every method on the data sets with an alphabet size of 4 and when varying the amount of states and splits of the original DRTA

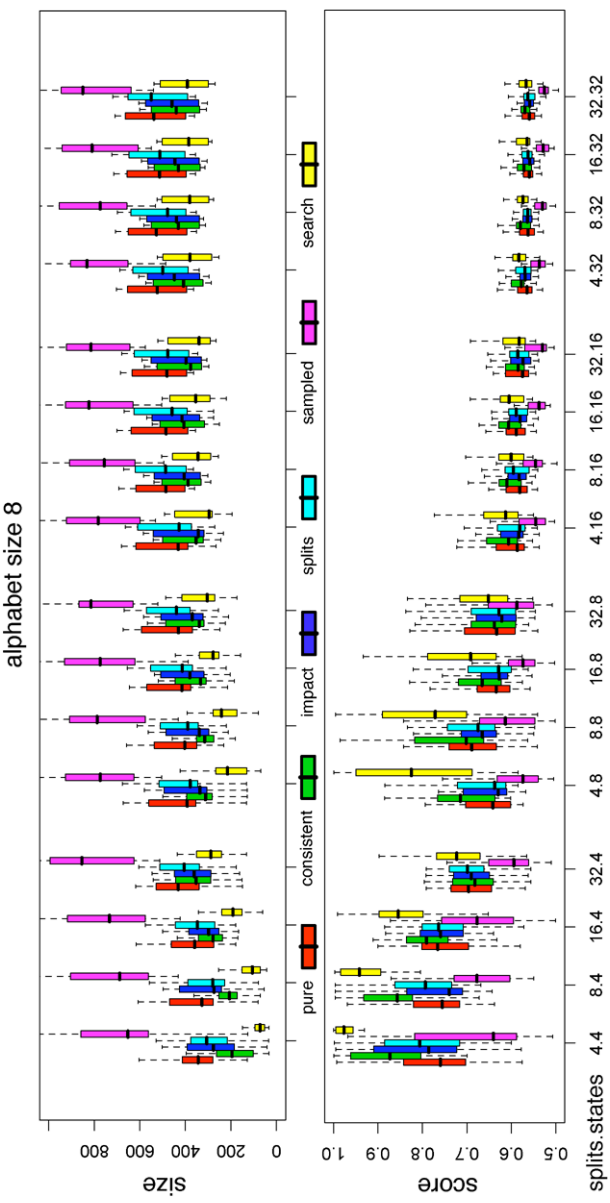

Fig. 16 Boxplots of the sizes and scores of every method on the data sets with an alphabet size of 8 and when varying the amount of states and splits of the original DRTA

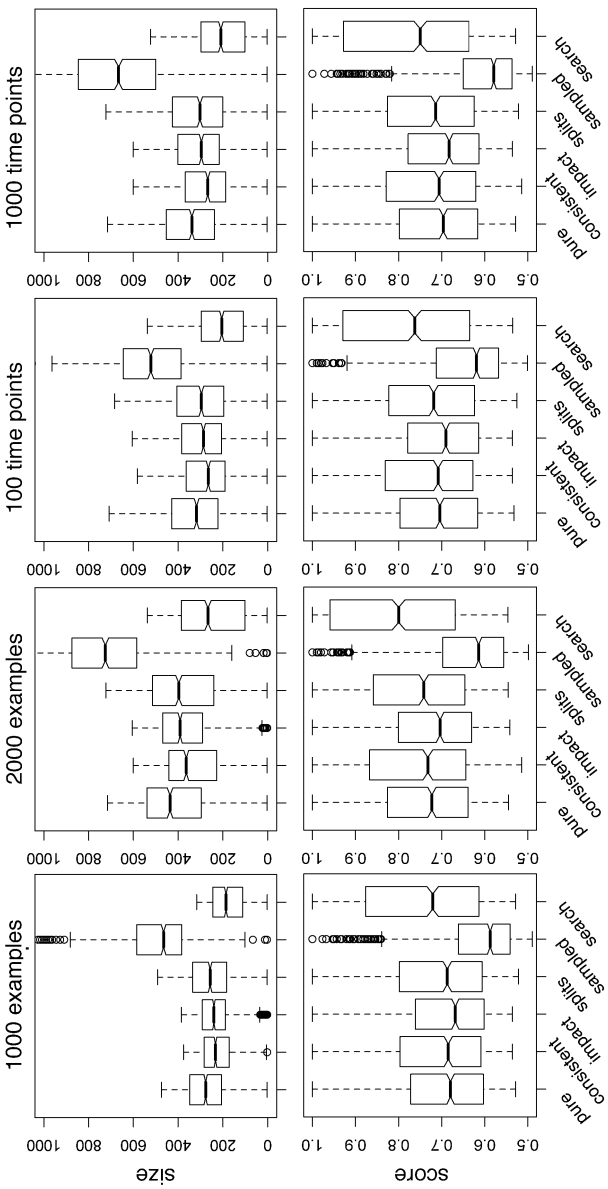

Fig. 17 Boxplots of the sizes and scores of every method when varying the amount of possible time values and the size of the data sets

– An increase in the number of time points does not affect the results of the DRTA identification algorithms much, but the performance of the sampling approach decreases significantly.

The first conclusion is very interesting since it seems to contradict the expectation that smaller DRTAs lead to better performance. However, as we can see from the plots, searching still helps, i.e., it decreases the size and increases the score. Thus, there seems to be a dependence on the sizes of DRTAs that perform well and the size of the data sample. Intuitively, the availability of more data makes it possible to infer more transitions correctly, even (or especially) in larger DRTAs. But this does not explain why RTI produces larger DRTAs when more data is available. We believe this occurs because more data also creates more possible inconsistencies (a positive example that ends in the same state as a negative example). To solve these inconsistencies, RTI requires larger DRTAs. This especially holds if RTI makes a (small) mistake, i.e., it performs an incorrect merge or split. In this case, when there is more data, more timed strings will end up in wrong states. Hence, more possible inconsistencies are created. Note that this phenomenon is not the same as the overtraining problem in machine learning, see e.g. Mitchell (1997). Overtraining occurs when a large number of parameters (states/splits) are fitted too closely to the data set. In such a case, the performance on the test set decreases. In our case, this performance increases.

The second conclusion is also interesting. That the sampling method performs worse is not very surprising since more time points results in a larger blowup. In addition, longer strings contain less useful data (it can create less inconsistencies) for the DFA identification algorithm, and hence the performance decreases. More interesting is the fact that it does not seem to matter much for RTI. In the DRTA case, an increase in the number of time points will make more splits possible, and hence results in a larger search space. When the amount of possible splits is larger, it makes sense that the chance that RTI chooses an incorrect split increases, and hence that the score decreases. Fortunately, in the boxplots of Fig. 17 this does not seem to occur.

Timing We did not perform any time measurements of our experiments. However, the consistent value can be computed very quickly. Running RTI with the consistent value over all of the 1920 problems took a few hours. Running RTI with the splits value took twice as long. The sampling approach took even longer, it took about 5 days in order to compute the results. Naturally, the search procedure took the longest, it required about 3 weeks. However, searching the smaller DRTAs (alphabet 4 and 8 states) usually finishes in one or two minutes. We performed the experiments on a 1.8 GHz PowerPC G5 computer.

7 Related work

As mentioned in the introduction, the most closely related work deals with the problem of learning event recording automata (ERAs) (Grinchtein et al. 2006). That work proposes an algorithm for learning these TAs from a timed teacher using membership and equivalence queries. It is possible to adapt a query learning algorithm to the setting of learning from a data set by simulating the queries using the data set (Goldman and Mathias 1996). Therefore, it should be possible to adapt the ERA query learning algorithm to the problem of learning an ERA from a data set. However, since the ERA learning algorithm requires an exponential amount of queries, this simulation would require a very large amount of data.

Other approaches to the problem of learning a timed system mainly deal with the inference of probabilistic timed models. One may observe that DRTAs are somewhat similar to

continuous time Markov models. The difference being (besides the transition probabilities) that these Markov models use a distribution over the execution times of events instead of time constraints. However, it may well be the case that a probabilistic variant of a TA can be used to define exactly the same timed probabilistic languages as some continuous-time hidden semi-Markov model, as is the case for regular probabilistic DFAs and hidden Markov models (Dupont et al. 2005).

For continuous-time Markov chains an identification method has been constructed (Sen et al. 2004). This method is based on a method to infer probabilistic DFAs, known as ALERGIA (Carrasco and Oncina 1994). The ALERGIA method is adapted by adding time distributions to the model and inference methods for these distributions to the algorithm. We believe continuous-time Markov chains to be too restrictive for our problem because the time values must be exponentially distributed. However, our approach is similar in that we adapt a known method (state-merging) to a timed setting and supply it with a specialized inference mechanism (splitting) for the timed properties of the model (a DRTA). For (hidden) semi-Markov models several inference methods exists (see, e.g., Guédon 2003). However, to the best of our knowledge these methods deal with the problem of optimizing the parameters of known models as to maximize the likelihood. Since in our problem we do not know the structure of the model to be identified, we are interested in methods that identify the structure in addition to the parameters.

In timed automata research, the model identification problem has not received a lot of attention. However, there are several related problems that can benefit from a good model identification method. For instance, the problem of testing timed automata (Springintveld et al. 2001) deals with a similar setting. Given access to a black-box system, the problem is to determine whether the system acts conform its specification. This problem is usually solved by finding a (preferably small) test set of labeled examples such that the system acts conform its specification if and only if it gives the correct labels to each of the examples. Given a model identification method and a set of labeled historic data, this problem can be solved in a different manner. The idea is to first identify a model from the historic data and then subsequently test this model either by hand (visually) or using a model verification tool, such as UPPAAL (Larsen et al. 1997).

Another closely related problem is that of controller synthesis for timed automata (Pnueli et al. 1998). Here the idea is to find a controller, that restricts the transitions a given TA is allowed to take, such that the behavior of the combined system satisfies certain properties. With the help of a model identification method, it becomes possible to synthesize controllers for unknown or black-box systems. This can be of importance in for example agent-based systems, where the internal structure of agents should remain unknown to other agents, but the behavior of the complete agent system should satisfy some properties, such as deadlock-freeness.

Closely related research comes from the temporal data-mining field (Roddick and Spiliopoulou 2002). In temporal data mining, the objective is to discover previously unknown rules from time series data. Moreover, these rules should be easy to understand and to validate. This is very much like the problem we are trying to solve. Our method for solving this problem looks very similar to one that is used to mine a hierarchy of temporal patterns from (multivariate) time series data (Mörchen and Ultsch 2004). It is different, however, in that this approach (and most other approaches in temporal data mining) focuses on finding simple patterns or correlations in the data, and not on finding a model for the actual system that generated this data.

8 Discussion

We have constructed a novel identification algorithm for deterministic real-time automata (DRTAs). These automata can be used to model systems for which the time between consecutive events is important for the system's behavior. We have adapted the state-merging algorithm for the identification of deterministic finite automata to the setting of real-time automata. To the best of our knowledge, ours is the first algorithm that can identify a timed automaton model from a timed input sample. Our results show that learning a DRTA returns a model of much higher accuracy than a straightforward adaptation of a state-of-the-art state-merging method by using sampling.

We believe that the main reason why RTI outperforms the sampling method is that RTI uses a heuristic to determine the values of the delay guards. In the sampling case, these values are fixed and lead to an exponential blowup. Intuitively, the use of a heuristic to avoid this blowup seems a good idea.

With RTI, we have developed an algorithm that can be used to automatically construct timed automaton models for real-time systems from data. Like many other commonly used models, the timed automaton models can be used to classify the behavior of such systems. In addition, however, a timed automaton model can also be used to analyze properties of the real-time system by applying for instance model checking. In this way, the RTI algorithm can prove to be very useful for providing insight into black-box real-time systems. It would be very interesting to try such a combination of techniques in future work.

For many applications, a problem of RTI can be that it requires *labeled* data. In many settings, it is very difficult to obtain labeled data: we can use sensors to observe systems, but not to label their behavior. Because we also want to apply RTI to such a setting, we will extend the RTI algorithm to the setting of unlabeled (only positive) data in future work. Such an adaptation can be made by using statistics as an evidence value. For the EDSM algorithm similar adaptations have been made, see, e.g., Kermorvant and Dupont (2002). In such an algorithm, two states cannot be merged if the strings that end up in those states are statistically different. In our case, the only difference is that RTI not only needs to decide whether two states are statistically different (should not be merged), but also whether they are statistically the same (should not be split). In addition, a trade-off will sometimes need to be made between these two alternatives.

Another possible issue can be that RTI identifies a restricted class of timed automata. It should not be too difficult, however, to adapt RTI in order to identify more general timed automata. This can be done for example by simply trying to reset clocks on the transitions to blue states. The automaton changes because of this reset and hence we can compute some evidence for this reset. We then only reset a clock if there is sufficient evidence for it. Since timed automata with resets can be very complex, it is unclear what the performance will be like. Furthermore, in the previous work (Verwer et al. 2008) we showed that in order to identify a timed automaton with two or more clocks, one will sometimes require an exponential amount of data. We therefore expect the performance of an adapted version of RTI (or any other algorithm) to perform poorly on identifying timed automata with two or more clocks. However, it would still be interesting to test this experimentally in future work.

Acknowledgements We thank anonymous reviewers for their helpful comments and suggestions on earlier versions of this article.

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

References

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

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Berlin: Springer.

Bugalho, M., & Oliveira, A. L. (2005). Inference of regular languages using state merging algorithms with search. *Pattern Recognition*, *38*, 1457–1467.

Carrasco, R., & Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method. In *LNCS: Vol. 862*. *Proceedings of the 2nd international colloqium on grammatical inference* (pp. 139–150). Berlin: Springer.

Clark, A., & Thollard, F. (2004) PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 473–497.

Dima, C. (2001). Real-time automata. *Journal of Automata, Languages and Combinatorics*, *6*(1), 2–23.

Dupont, P., Denis, F., & Esposito, Y. (2005). Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, *38*, 1349–1371.

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

Goldman, S. A., & Mathias, H. D. (1996). Teaching a smarter learner. *Journal of Computer and System Sciences*, *52*(2), 255–267.

Grinchtein, O., Jonsson, B., & Petterson, P. (2006). Inference of event-recording automata using timed decision trees. In *LNCS: Vol. 4137*. *CONCUR* (pp. 435–449). Berlin: Springer.

Guédon, Y. (2003). Estimating hidden semi-Markov chains from discrete sequences. *Journal of Computational and Graphical Statistics*, *12*(3), 604–639.

de la Higuera, C. (1997). Characteristic sets for polynomial grammatical inference. *Machine Learning*, *27*(2), 125–138.

de la Higuera, C. (2005). A bibliographical study of grammatical inference. *Pattern Recognition*, *38*(9), 1332–1348.

Kermorvant, C., & Dupont, P. (2002). Stochastic grammatical inference with multinomial tests. In *LNAI: Vol. 2484*. *Proceedings of the 6th international colloquium on grammatical inference* (pp. 149–160). Berlin: Springer.

Lang, K. J., Pearlmutter, B. A., & Price, R. A. (1998). Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *LNCS: Vol. 1433*. *Grammatical inference*. Berlin: Springer.

Larsen, K. G., Petterson, P., & Yi, W. (1997). Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, *1*(1–2), 134–152.

Mitchell, T. (1997). *Machine learning*. New York: McGraw-Hill.

Mörchen, F., & Ultsch, A. (2004). Mining temporal patterns in multivariate time series. In *LNCS: Vol. 3238*. *Advances in artificial intelligence* (pp. 127–140). Berlin: Springer.

Oncina, J., & Garcia, P. (1992). Inferring regular languages in polynomial update time. In *Series in machine perception and artificial intelligence: Vol. 1*. *Pattern recognition and image analysis* (pp. 49–61). Singapore: World Scientific.

Pitt, L., & Warmuth, M. (1989). The minimum consistent DFA problem cannot be approximated within and polynomial. In *Annual ACM symposium on theory of computing* (pp. 421–432). New York: ACM.

Pnueli, A., Asarin, E., Maler, O., & Sifakis, J. (1998). Controller synthesis for timed automata. In *IFAC symposium on system structure and control* (pp. 469–474). Amsterdam: Elsevier.

Roddick, J. F., & Spiliopoulou, M. (2002). A survey of temporal knowledge discovery paradigms and methods. *IEEE Transactions on Knowledge and Data Engineering*, *14*(4), 750–767.

Sen, K., Viswanathan, M., & Agha, G. (2004). Learning continuous time Markov chains from sample executions. In *Proceedings of the quantitative evaluation of systems* (pp. 146–155).

Sipser, M. (1997). *Introduction to the theory of computation*. Boston: PWS Publishing.

Springintveld, J., Vaandrager, F. W., & D'Argenio, P. R. (2001). Testing timed automata. *Theoretical Computer Science*, *254*(1–2), 225–257.

Sudkamp, T. A. (2006). *Languages and machines: an introduction to the theory of computer science* (3rd ed.). Reading: Addison-Wesley.

Verwer, S., de Weerdt, M., & Witteveen, C. (2008). Polynomial distinguishability of timed automata. In *LNCS: Vol. 5278*. *Grammatical inference: theory and applications* (pp. 238–251). Berlin: Springer.

Verwer, S., de Weerdt, M., & Witteveen, C. (2009). One-clock deterministic timed automata are efficiently identifiable in the limit. In *LNCS: Vol. 5457*. *Language and automata theory and applications* (pp. 740–751). Berlin: Springer.

Verwer, S., de Weerdt, M., & Witteveen, C. (2011). The efficiency of identifying timed automata and the power of clocks. *Information and Computation*, *209*(3), 606–625.