Learning Probabilistc Real Time Automata From Multi-Attribute Event Logs

Jana Schmidt Stefan Kramer

TUM-I11
Oktober 11

This documents presents the notation and formalisms for probabilistic real time automata (PRTAs). The full paper can be found in the journal:

Learning Probabilistic Real-Time Automata from Multi-Attribute Event Logs
Intelligent Data Analysis, special issue on Dynamic Networks and Knowledge Discovery. Vol. 17(1), 2013. IOS Press. [5]

Contents

Abstract

The growing number of time-labeled datasets in science and industry increases the need for algorithms that automatically induce process models. Existing methods are capable of identifying process models that typically only work on single attribute events. We propose a new model type and its corresponding algorithm to address the problem of mining multi-attribute events, meaning that each event is described by a vector of attributes. The model is based on timed automata, includes expressive descriptions of states and can be used for making predictions. A probabilistic real time automaton (PRTA) is created, where each state is annotated by a profile of events. To identify the states of the automaton, similar events are combined by a clustering approach. The method was implemented and tested on a synthetic, a medical and a biological dataset. Its prediction accuracy was evaluated on a medical dataset and compared to a combined logistic regression, which is considered a standard in this application domain. Moreover, the method was experimentally compared to Multi-Output HMMs and Petri nets learned by standard process mining algorithms. The experimental comparison suggests that the automaton-based approach performs favorably in several dimensions. Most importantly, we show that meaningful medical and biological process knowledge can be extracted from such automata.

1 Introduction

Recent years have seen a surge of interest in temporal data in all areas of science and industry. In the field of molecular biology and medicine, for instance, data sets of time-labeled observations may provide insights into cellular processes or the progression of disease. Typical data include the description of stages (or states) as well as their temporal relation, which has to be captured by suitable models. One possible way of representing complex temporal phenomena is by timed automata [1], which are finite state models explicitly modeling time. They can be linked to domain concepts and help to reason about real time processes. Until now, experts construct such models by hand, which can be time consuming and error prone. The situation is even more complicated if the states of the process are described by multiple attributes. In fact, in such a setting even the definition of a state is unclear. To deal with the problem of automatic extraction of meaningful, expressive and temporally ordered states, we propose a new algorithm based on finding real-time automata. Observations consist of a multi-dimensional attribute vector and a corresponding time point, denoting when the state was observed. The implicit modeling of time, e.g., by an untimed model like hidden Markov models (HMMs) would result in a combinatorial explosion of states. The same is true for modeling multiple state characteristics. We solve this problem by adding profiles to states that represent all their events and the states' characteristics. The annotation of states makes the problem feasible and additionally the resulting model easier to understand.

This report is organized as follows: First, a formal definition of the model and a description of the algorithm for learning probabilistic real time automata (section 3) is presented. Subsequently, the workflow and how the state merging is conducted, will be discussed. The report closes with an overall conclusion.

2 Probabilistic Real Time Automata (PRTA)

In this section, we present an algorithm for learning probabilistic real-time automata (PRTAs) which is, like the currently best method for learning automata [3], based on state merging in a prefix tree. Our type of automaton models a discrete event system (DES) [6]. Let dataset D of instances I be given $D = \{I_1, \dots, I_n\}$, where each instance I_i represents a sequence of timed events: $I_i = (\vec{e}_1, t_1)(\vec{e}_2, t_2) \dots (\vec{e}_k, t_k)$. This event sequence, ordered by the time of occurence, is called a *history*. An event e_i is a binary vector $\vec{e}_i = (a_{i1}, \dots, a_{im})$ that specifies whether attribute a_{ij} is observed ($a_{ij} = 1$) in this event. Because the events e_i have a time stamp t_i assigned, a timed language model is created. Each event $(\vec{e}_i, t_i)$ can also be described by a conjunction of all attributes that are present at time t_i[1]. Let $|\vec{e}_i|$ denote the number of attributes equal to one in this event. Every time-stamp value $t_i \in \mathbb{N}$ represents the time that has elapsed since the previous event of the instance has occurred. A *PRTA* is a directed graph with states Q and transitions T. Each state q_i holds its set of events E_i and is – for simplicity of notation – annotated by a so-called *profile* f_i. The profile shows the mean attribute/feature vector of all events that are mapped to q_i:

[1] This is similar to an itemset representation.

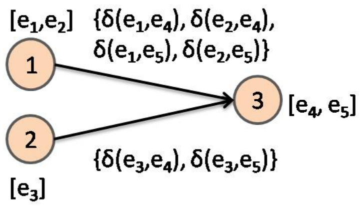

Figure 1: Event annotation of transitions and states. Each state's profile consists of the events that are incorporated in the state. Each transition from state q_i to q_j is labeled by the difference of the events that were observed between states q_i and q_j.

$$\vec{f}_i = \frac{\sum_{e \in E_i} e}{|E_i|}. \tag{1}$$

In other words, a profile is just a summary of these events. Transitions $t_{ij} \in T$ of the PRTA connect two states q_i and q_j and are annotated with a delay guard to reflect the observed time intervals between the connected states. A delay guard is defined as an interval $[t_1, t_2]$ with $t_1, t_2 \in \mathbb{N}$, where $t_1 (t_2)$ defines the minimal (maximal) number of time steps when this transition can be passed. Additionally, transitions $t_{i,j}$ of the PRTA are labeled with the set $T_{L_{i,j}}$ that describes the changes of the profiles from state q_i to q_j. These changes are expressed in the so-called *delta notation*: $T_{L_{i,j}} = \Delta(E_i, E_j)$, where

$$\Delta(E_i, E_j) = \bigcup_{\vec{e}_k \in E_i, \vec{e}_l \in E_j} \delta(\vec{e}_k, \vec{e}_l) \tag{2}$$

and $\delta(\vec{e}_k, \vec{e}_l)$ is defined as the difference of the binary vectors $\vec{e}_k$ und $\vec{e}_l$: $\delta(\vec{e}_k, \vec{e}_l) = \vec{e}_k - \vec{e}_l$. Thus, the label is the set of differences between the elements of E_i and E_j and can be interpreted as the set of change vectors that are necessary to reach q_j from q_i. Figure 1 gives an example of how transitions and states incorporate the events and the corresponding differences. To complete the transition, it has assigned a probability $p_{i,j}$ of occurrence. The sum of all probabilities of outgoing transitions of a state is equal to one. In general, automata have a set of start (S) and final states (F) that are a subset of all states in the automaton ($S \subset Q \wedge F \subset Q$). In a PRTA, $S = Q$ and $F = Q$, because each state is allowed to be a start or final state. A PRTA is then formally defined as follows:

Definition 1 *A PRTA Γ is a tuple $\Gamma = (Q, \sum, T, S, F)$, where*

- *Q is a finite set of states*
- *Σ is a finite set of events to label the transitions*
- *T is a finite set of transitions*
- *$S = Q$ is the set of start states*
- *$F = Q$ is the set of final states*

A state $q_i \in Q$ is a pair $\langle E_i, \vec{f}_i \rangle$ where E_i is its set of events ($E_i = \{\vec{e}_k : \vec{e}_k \in C_i\}$) [2] and $\vec{f}_i$ is an attribute vector called its profile. Σ are all events $\vec{e}$ that are observed in the input data. A transition $t \in T$ is a tuple $\langle q, q', T_L, \phi, p \rangle$ where $q, q' \in Q$ are the source and target states, $T_L = \Delta(E_i, E_j)$ and ϕ is a delay guard defined by an interval $[t_1, t_2]$ with $t_1, t_2 \in \mathbb{N}$. p defines a probability $p \in [0, 1]$ that this transition occurs.

[2] C_i is a cluster of events and will be described in more detail in Section 2.2

2.1 Accepting words

One task of automata is to decide whether they accept a given word of a language. This section will describe how this problem can be solved for a PRTA using the Δ-notation.

2.1.1 Definition of words

Let $\Sigma = \{\vec{e}_1, \ldots, \vec{e}_k\}$ be an alphabet over binary vectors. Then $L \subseteq (\Sigma, \mathbb{N})^+$ is a language over pairs of the alphabet Σ and time points of $\mathbb{N}$, and $(\vec{e}_i, t_j)(\vec{e}_k, t_l)\ldots(\vec{e}_m, t_n)$ is a word with time labels from L. In such languages, a time point t_i denotes when its corresponding element $\vec{e}_i$ has occurred. Timepoints are given relatively, i.e., each time point reflects the time that has elapsed since the last event. As an example, consider language

$$L = \{(\vec{e}_1, 4)(\vec{e}_4, 2)(\vec{e}_{18}, 1),\ (\vec{e}_2, 3)(\vec{e}_1, 6)(\vec{e}_5, 3),\ (\vec{e}_9, 10)\}$$

Further, let the Δ-notation for a word $w = (\vec{e}_i, t_j)\ldots(\vec{e}_m, t_n)$ be given by

$$\Delta(w) = (\delta(\vec{e}_1, \vec{e}_2), t_2)(\delta(\vec{e}_2, \vec{e}_3), t_3), \ldots, (\delta(\vec{e}_{m-1}, \vec{e}_m), t_m) \tag{3}$$

Again, it shows the difference from one event to the next, and the time that has elapsed. The problem of deciding whether an automaton Γ accepts a word w (if w is part of the language L modeled by Γ) is transformed into a check if there exists a valid sequence $G = q_0 T_1 T_2 \ldots T_{m-2} T_{m-1}$ of transitions, which comprises the word w in Δ-notation.

$$(\delta(\vec{e}_i, \vec{e}_{i+1}), t_{i+1}) \in (L_{T_i}, \phi_i) \quad \leftrightarrow \quad \delta(\vec{e}_i, \vec{e}_{i+1}) \in L_{T_i} \wedge t_{i+1} \in \phi_i$$

The first transition must leave state q_0 that represents w_1: $w_1 \in E_0$. A sequence is valid if and only if succeeding transitions share states (are adjacent):

$$\forall\ T_i, T_{i+1} \in G : Target(T_i) = Source(T_{i+1}) \tag{4}$$

$Source(T_i)$ ($Target(T_i)$) names the source (target) state of a transition T_i.

2.1.2 Solving the word problem

The language of a PRTA is given by: $L_\mu = \{w \in (\Sigma, \mathbb{N})^+ \mid q_0 P_w Q_{accept} > \mu\}$, where μ is a probability threshold and Q_{accept} is the set of all final states. P_w describes the probability for a word w and a state sequence $(q_1, \ldots, q_m)$:

$$P_w = \prod_i^m ((q_i, q_{i+1}, \delta(\vec{e}_i, \vec{e}_{i+1}), \phi_i, p_i)) \tag{5}$$

In the case of a PRTA $Q = Q_{accept}$ and $\mu = 0$, i.e., there must exists a path leaving from q_0 (with a joint probability greater than zero) that ‘consumes’ word w. State q_0 is exactly the state that represents $\vec{e}_1$: $\vec{e}_1 \in E_0$. For this problem, it is easy to give an algorithm that terminates after a finite number of steps. Algorithm 1 shows the initialization of the problem (finding the initial state q_0) and then calls the search for a transition sequence. Algorithm 2 describes this search[3] and the stopping criterion. If there is no edge with the required label, the automaton does not accept the word. In contrast, if the ‘last’ edge is found, the automaton accepts the word and returns the probability of the word.

[3] $|w|$ gives the length of the word w and $w[x]$ the xth event of word w.

Algorithm 1 ParseWord (PRTA Γ, ArrayList w)

```
$q_0$ = findFirstState(PRTA, w[1])
if $q_0$ is not null then
    accept = ParseRemainingWord($q_0$, w, 1 )
else
    return 0
end if
return accept
```

Algorithm 2 ParseRemainingWord (State q, ArrayList w)

```
$T$ = TransitionsWithGivenDeltaAndTimeLabel($q$, $\delta(w[1], w[2]), t_2$ )
if $T = \emptyset$ then
    return 0
end if
$w = w \backslash w[1]$
if $|w| == 0$ then
    return 1
end if
return $p(t) \times$ ParseRemainingWord ($Target(t)$, w )
```

2.2 Induction of a PRTA

In the following, we describe how PRTAs can be learned. The top-level algorithm is shown in Algorithm 3. As input, the algorithm expects a finite set of histories. From the

Algorithm 3 InducePRTA (Histories H, Parameter *params*)

```
$prefixTree \leftarrow$ createPrefixTree($H$)
$M \leftarrow$ calculateDistanceMatrix($prefixTree$)
$res \leftarrow$ cluster($M$, $params$)
while $res \neq \{\}$ do
    C $\leftarrow$ getNextCluster($res$)
    $prefixTree =$ mergeStatesInPrefixTree($C$)
    $res \leftarrow$ deleteFromResult($C$)
end while
computeQualityMeasure($prefixTree$)
return $prefixTree$
```

histories, the algorithm first constructs a prefix tree acceptor (PTA). A PTA is a PRTA in the form of a tree in which exactly one path exists to any state. Each leaf represents one or more instances from the input set. If input histories have the same prefix, then they share the path of this prefix, while the suffix has its own path. When a new history is put in the PTA, there are the following possibilities:

1. No prefix of the history is represented by an existing path in the PTA.
2. There is a path that represents a prefix of the history.
3. There is a path that represents the whole history.

In the first case, a new path starting at the root from the prefix tree is inserted in the PTA. The probabilities of all transitions on the path are set to one, and all delay guards are set to the current time stamp. In the second case (if the PTA shares a prefix with the history), all probabilities on the equivalent path are updated corresponding to the annotated probabilities. Consider a transition from state q_i to q_j on the prefix path of the history and assume that q_i has k other outgoing transitions. For the transition that is covered by the new history, the according probability p' is updated to $p' = \frac{|q_j|+1}{|q_i|+1}$ where $|q|$ denotes the frequency of a state. This ratio actually is the maximum likelihood estimation $b_{ij} = P(transition_{ij}|\vec{e}_k, t_k)$. For all remaining outgoing transitions $t_l(l = 1, \ldots, k)$ of state q_i, the probability p'_l is recomputed by $p'_l = \frac{|q_l|}{|q_i|+1}$. If the time constraint t of the history does not meet the time constraint of the existing transition, the delay guard ϕ'_k is expanded so that it includes the new time constraint: $\phi'_k = [a, b]$, where $a = min(\phi_k, \phi_l)$ and $b = max(\phi_k, \phi_l)$. If the end of the path that represents the shared prefix is reached, a new path with the remaining events of the history is appended, following the description of case one. In the third case, all transition probabilities and delay guards are updated as described for the second case, but no additional path is added to the PTA (this procedure is also referred to as determinization). After creating the PTA with all input histories, the goal is to produce a PRTA that is minimal. Minimal means that a minimum number of states should be derived, but reflecting a maximum of information. This condition is heuristically motivated by $Occam's\ Razor$. The parameter that leads to the minimal model is usually given by the user, in our case, a certain distance threshold between mergeable states. To obtain a compact model, merges of nodes in the prefix tree are performed. A merge is an operation where two states q_i and q_j are combined into one new state q_k. Because homogeneous states shall be identified, clustering is applied. In general, a merge step is the aggregation of all states belonging to a cluster into one new state with a new profile. A merge combines all profiles $\vec{f_i}$ of the states q_i to be merged into one single profile $\vec{f_k}$ by their weighted mean:

$$\vec{f_k} = \frac{1}{\sum_{q_i \in C_k} |E_i|} \sum_{q_i \in C_k} |E_i| \times \vec{f_i} \tag{6}$$

Which states are to be merged is identified via clustering. Therefore, a cluster assignment for each state in the prefix tree must be found.[4] In general, the input for a cluster algorithm is a distance matrix (or a distance function and the instances respectively). However, when constructing an automaton, the input for the clustering is a prefix tree. During the clustering, each state of the prefix tree is handled as an individual instance. The attributes of the instance are the values of the state's histogram (cf. equation 1). They can be used as a basis for the computation of distances between states. Let us consider the clustering as a function c (cf. equation 7) that maps each state q to a cluster identifier $k \in \mathbb{N}$.

$$c(q) : Q \to \{1, \ldots, k\} \tag{7}$$

Then it is possible to evaluate each possible clustering $c_i(q)$ with some quality function G (consider, e.g., the silhouette coefficient or an optimal inter/intra cluster distance).

[4] In general, the order of PTA construction and clustering is irrelevant, they are just required before the state merging is started.

The result of the clustering algorithm is the mapping $c(q)^*$ that maximizes the quality function.

$$c(q)^* = argmax_i \; G(c_i(q)) \tag{8}$$

Note that depending on the application domain, the user can decide which distance function, clustering algorithm and quality function is best suited. By using the best function $c(q)^*$, the merge procedure creates for each cluster identifier k one new state in the prefix tree by merging all states q which are mapped to cluster k. Formally, the inverse function $c^{-1}(q)^*$ returns for each cluster identifier k the set of states that are mapped to it. The automaton is created by merging all states q of clusters k one after the other. To preserve consistency, update operations on transitions have to be performed. If two states q_i and q_j are to be merged and there are no transitions t_k where $t_k = \langle q_k, q'_k, T_{L_k}, \phi_k, p_k \rangle$ and $t_l = \langle q_l, q'_l, T_{L_l}, \phi_l, p_l \rangle$ with $q_k = q_l, q'_k = q_i$ and $q'_l = q_j$ (they do not share a predecessor), change t_k to $\langle q_l, q'_k, T_{L_l}, \phi_l, p_l \rangle$ (re-link the transition) and compute the new profile of q_i using equation 1. q_j can be deleted from the prefix tree. If there exist two transitions t_k and t_l with $q'_k = q_i \wedge q'_l = q_j \wedge q_k = q_l$ (they share the same start but not end state), the transitions have to be merged additionally. This means that $\langle q_k, q'_k, T_{L_k}, \phi_k, p_k \rangle$ is to be updated to $\langle q_k, q'_k, T_{L_k}, \phi'_k, p_k + p_l \rangle$. $\phi'_k = [a,b]$, where $a = min(\phi_k, \phi_l)$ and $b = max(\phi_k, \phi_l)$.[5] The updated probability p' is again calculated with the counts of the states $p' = \frac{|q'_k|+|q'_l|}{|q_k|} = p_k + p_l$. Note that the labels of the transitions are not updated until the end of the merge procedure. Then, each state holds its set of events and the labels T_{L_k} can be easily computed by calculating the difference between the two sets E_i and E_j. Algorithm 4 shows this procedure.

Algorithm 4 CreateTransitionLabels (q_i, q_j)

```
for each $e_k \in E_i$ do
    for each $e_l \in E_j$ do
        add $\delta(e_k, e_l)$ to labels of transition $t(q_i, q_j)$
    end for
end for
```

One additional property of PRTAs is that a label $\delta_l \in T_{L_k}$ is only present on exactly one outgoing transition of a state q_i. Given that the underlying distance based clustering optimizes the inter and intra cluster distance, respectively, it can be shown that $\nexists\, \delta_l \; : \; \delta_l \in T_{L_k} \wedge \delta_l \in T_{L_m}$. The proof works by contradiction: consider the case where there exist three states $q_0 = \langle \{x_1, x_2\}, f_0 \rangle$, $q_1 = \langle \{y_1\}, f_1 \rangle$ and $q_2 = \langle \{y_2\}, f_2 \rangle$, with the transitions $t_1 = \langle q_0, q_1, \delta_l, \phi_1, p_1 \rangle$ and $t_2 = \langle q_0, q_2, \delta_l, \phi_2, p_2 \rangle$, i.e. two transitions, each holding label δ_l. Moreover, let δ_l be defined as $\delta(x_1, y_1)$, $y_1 \notin E_2$ and $y_2 \notin E_1$. Let us further assume, without loss of generality, that δ_l consists of three consecutive blocks, e.g., $\delta_l = (+1 + 1 - 1 - 1000)$. Let k be the first position of the 0-block at the end. Following the assumption that δ_l exists on both transitions, we can specify constraints for x_2 and y_2. First, $x_{1_j} := x_{2_j}$ and $y_{1_j} := y_{2_j}$ for all $j < k$, because there is only one possibility for $y_j - x_j$ to be equal to -1 (and 1) in a binary setting. Second, $y_{2_j} := x_{2_j}$ for $j \geq k$, because otherwise δ_{l_j} is not 0. Additionally, there must exists at least one

[5] In other words, we conduct the least general generalization on the time intervals to be merged.

$x_{1_j} \neq x_{2_j}$ to ensure that $y_2 \neq y_1$. Otherwise, y_1 and y_2 are equal and thus clustered together, so t_2 cannot exist. With these constraints: (1) $x_{1j} = x_{2j}$, $y_{1j} = y_{2j}$ $(i < k)$, (2) $y_{2_j} := x_{2_j}$ $(i \geq k)$ and (3) $y_1 \neq y_2$ the distance $d(x_1, y_1)$ reveals to be equal to $d(x_2, y_2)$, because the distance for parts $i \leq k$ is 0 (2) and the remaining distances are equal (1). As y_1 and y_2 must reside in different clusters, the clustering then returns a solution where pairs of objects with the same distance are clustered together once, and not a second time. This is inconsistent and also shows that the inter and intra cluster distance in this clustering cannot be optimal. This leads to the conclusion that for all suitable distance based clusterings there is no δ_l that occurs on more than one outgoing transition of one state q_0. Finally, this conclusion even allows to check each learned automaton whether the intra cluster distances are above a threshold to ensure that each δ_l occurs only once.
The delay guard generalization is motivated by the use of positive instances only. Remember that all input instances shall be accepted by the automaton, indicating that they are positive. Furthermore, we assume that when no continuous timeframe is present in the data, it is not because it does not exist, but because of lack of data.[6] However, the profiles of the states q_i and q_j have to be updated with their weighted mean. Transition t_l is deleted. The same operations have to be applied on all outgoing transitions of states q_i and q_j. After all merges are conducted, a quality measure of the clustering is computed. Like in every clustering problem, the proportion of inter- and intra-cluster distances is of interest. The silhouette coefficient (SC) [4] evaluates this proportion independent of the number of resulting clusters and is a measure for the homogeneity of the states of the automaton. The SC of a state q_j representing a cluster C is calculated in the following way:

$$SC_C = \sum_{o \in C} \frac{b(o) - a(o)}{max\{a(o), b(o)\}} \tag{9}$$

$a(o)$ is the distance to the own cluster center, while $b(o)$ denotes the distance to the second next cluster center. The SC for the automaton is computed by averaging the SC for each state. It always holds that $-1 \leq SC \leq 1$. Good results are expected above an SC of 0.5. However, a high SC does not necessarily reflect the best clustering, since $SC_C = 1$ for all $|C| = 1$ or $|C| = X$ where X is the number of all instances. Generally, SC_C increases with smaller clusters, because $SC_C = 1$ for all states C that consist of one example only.
The clustering method can be different for each use case. We decided to cluster with a divisive hierarchical cluster algorithm because it enables us to define a distance threshold that has to be fulfilled by the clustering. Furthermore, correlations between states can be investigated, and upper and lower distance thresholds can be set according to domain-specific constraints. In contrast, it will not be possible to tell a priori how many states are to be expected. This is why k-means like clustering methods are not appropriate in our scenario. Diana [4] is a divisive hierarchical clustering algorithm which computes a final dendrogram that shows how 'related' the states are.[7] With this dendrogram and a specific distance constraint, the user can create a suitable clustering. The

[6]This assumption can be made in the medical application domain presented below. Here data is not recorded in regular time steps but whenever people go to the physician. So, there are gaps in the data recording process, which are taken into account by the above assumption.

[7]Preliminary experiments with a variety of other clustering algorithms like K-medoids [4], DBScan [2], EM, and Farthest First did not produce any usable results.

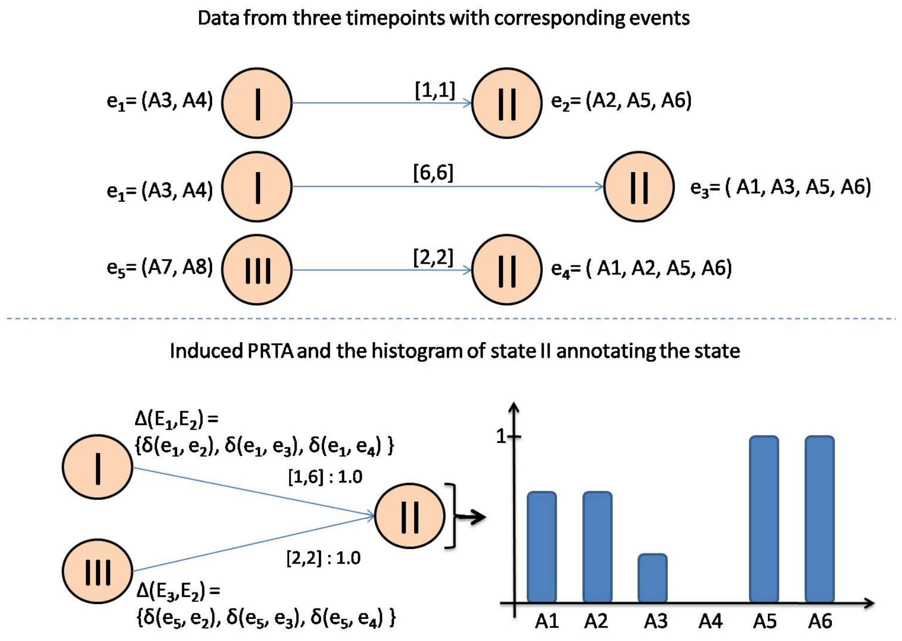
Data from three timepoints with corresponding events
e1= (A3, A4)
I
[1,1]
II
e2= (A2, A5, A6)
e1= (A3, A4)
I
[6,6]
II
e3= (A1, A3, A5, A6)
e5= (A7, A8)
III
[2,2]
II
e4= (A1, A2, A5, A6)
Induced PRTA and the histogram of state II annotating the state
Δ(E1,E2) =
{δ(e1, e2), δ(e1, e3), δ(e1, e4) }
[1,6] : 1.0
[2,2] : 1.0
Δ(E3,E2) =
{δ(e5, e2), δ(e5, e3), δ(e5, e4) }
1
A1 A2 A3 A4 A5 A6

Figure 2: Example creation of a PRTA

dendrogram is cut according to the distance constraint, and all objects that are still connected form a cluster. Nevertheless, one critical point for any clustering is to choose an appropriate distance measure. As this is domain-dependent, it will be discussed in the section on experimental results.
In Figure 2, a merge step during the construction of a PRTA is illustrated. We see here that the states are annotated with a profile and that transitions consist of all observed possibilities to end in a state, a path probability and a delay guard. The upper part of the figure displays three sample histories in a prefix tree before the merging starts. For simplicity, the root of the prefix tree is not shown. The number in the states indicates to which cluster the states belong to. Two of them are in cluster one, three are in cluster two and only one state in cluster three. On all transitions only one event that leads from the left to the right state and the delay guard is displayed. A delay guard of [1,1] means that the event followed exactly one time step after the first event. The annotation of $A2$, $A5$, $A6$ on the transitions means that these attributes were observed after this state. The lower part of the figure shows the automaton after the merging step (without the root). The transitions and delay guards are updated following the rules. The number behind the colon reflects the probability of this path. In this case, they are always equal to one because there exists no splitting transition.

2.3 Predicting with an automaton

In this section, we explain how such an automaton can be used to make predictions. With a PRTA, we cannot only map processes that are reflected in the data but also make predictions about how the next state of an instance will be. Of course, it is also possible to predict series of subsequent states. The task of the prediction for a new instance can be formalized as follows: Given an instance x denoted by its feature vector $\vec{f}_x = (f_1, f_2, \ldots, f_n)$, we want to identify the profile $\vec{f}^* = (f_1^*, f_2^*, \ldots, f_n^*)$ it will develop after l time steps. A prediction for an instance is done by first identifying the state in the PRTA that is most similar to the given event distribution of the instance. This is the start state q_{start} for the prediction:

$$q_{start} = argmin_{q_i} d(\vec{f}_x, \vec{f}_i) \tag{10}$$

Distance function d will be introduced in a subsequent section. Given an arbitrary state q, let $q_1, \ldots, q_k$ denote the states with incoming transitions from q, and $p_1, \ldots, p_k$ be the probabilities on those transitions. Moreover, let $[t_{1,1}, t_{1,2}]$ to $[t_{k,1}, t_{k,2}]$ denote the delay guards for the transitions. Then the predicted profile given l time steps starting with state q is defined as:

$$\begin{aligned} f^*(q,l) &= \sum_{t_{i,2}>l} p_i \times f_q + \\ & \quad \sum_{t_{i,2}\leq l} p_i \times f^*(q_i, l - t_{i,2}) \end{aligned}$$

The next state is predicted according to the transition probabilities and their delay guards. The first summand represents the case where state q is not left, because it consumes all the ‘remaining’ time. Thus, no other following state is considered for the prediction. The second summand represents the case where the state *has to be left*. In the latter case, this means that the predicted profile of the next state is used. If q does not have any outgoing transition, then $f^*(q,d) = f_q$, i.e. the profile on the state itself. To obtain a prediction for the test instance, we apply f^* to q_{start}. Note that we make the assumption that the automaton stays maximally long in a state, i.e., the transition is made as late as possible. Moreover, to leave a state, the delay guard has to meet the time constraint l: $l > [t_{k,1}, t_{k,2}]$. Then, the remaining time l' for the next steps is reduced by the maximum of the delay guard $l' = l - t_{i,2}$. These constraints lead to a definite prediction of the profile: There is only one possible solution for the prediction. Another assumption would be that a transition is made as early as possible. Again, one definite prediction is achieved. However, an arbitrary time consumption of each state could be desired as well. In this case, the prediction would be dependent on all (valid) possible time consumptions of each state. Accordingly, the prediction is the averaged profile of all resulting predictions. Using the later approach may result in an exponential number of possible results and a highly blurred predicted profile. That is why a definite prediction was chosen in this paper. However, the choice of a prediction constraint is likely to be highly domain-dependent. By calculating joint profiles, we can predict the change of attributes depending only on the instance's attribute setting. However, if one is interested in the development of a certain state, the automaton gives probabilities for future states. This is a main advantage of the automaton, because many comparable algorithms only give one prediction. Consider a system that has alternatives for each state (e.g., medical therapy options and outcomes or biochemical pathways), meaning that a certain percentage of instances will take either the one or the other path. So, a distribution of resulting states is the desired output. The automaton aims at modeling such alternatives and their corresponding probabilities.

3 Conclusion

In this paper, we proposed a new method for learning process models in the form of probabilistic real-time automata (PRTA) for multi-attribute event logs. To learn such models, a prefix tree is created, in which states are merged when similar enough. State merging is employed because it is currently the best method for learning finite automata in grammatical inference [3]. In order to identify states to be merged, a divisive hierarchical clustering method is used. The algorithm was evaluated on synthetic data, for which the true underlying process was known. Moreover, it was tested on real data from a medical and a biological application domain to examine the resulting structure. The experiments showed that the automaton can be related to domain knowledge. To compare the ability of structure identification, standard process mining algorithms [7, 8] were applied on the same synthetic data set. However, they were not able to reveal the correct structure of the underlying process, because of the additional constraints they have to fulfill. Finally, to evaluate the predictive power of the PRTA, the distance of the predicted profiles to the profiles of the true next states was computed. The predictions of the PRTA were compared to the predictions of a combined logistic regression approach, which is considered a standard in this application domain. Additionally, a Multi-Output HMM was trained and tested for its predictive power. The results suggest that the automaton-based prediction performs favorably compared to both logistic regression and Multi-Output HMMs.

References

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

[2] D. Arlia and M. Coppola. Experiments in parallel clustering with DBSCAN. In *Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, pages 326–331. Springer-Verlag, 2001.

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

[4] J. Kalbfleisch. *Probability and Statistical Inference: Vol. 2: Statistical Inference*. Springer, 1985.

[5] J. Schmidt, A. Ghorbani, A. Hapfelmeier, and S. Kramer. Learning probabilistic real time automata from multi attribute event logs. *Intelligent Data Analysis - Special Issue*, 7(1), 2013.

[6] S. E. Verwer, M. M. de Weerdt, and C. Witteveen. Identifying an automaton model for timed data. In Y. Saeys, E. Tsiporkova, B. D. Baets, and Y. van de Peer, editors, *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn)*, pages 57–64, 2006.

[7] L. Wen, J. Wang, and J. Sun. Detecting implicit dependencies between tasks from event logs. In X. Zhou, J. Li, H. Shen, M. Kitsuregawa, and Y. Zhang, editors, *Frontiers of WWW Research and Development - APWeb 2006*, volume 3841 of *Lecture Notes in Computer Science*, pages 591–603. Springer, 2006.

[8] L. Wen, J. Wang, and J. Sun. Mining invisible tasks from event logs. In G. Dong, X. Lin, W. Wang, Y. Yang, and J. Yu, editors, *Advances in Data and Web Management*, volume 4505 of *Lecture Notes in Computer Science*, pages 358–365. Springer, 2007.