JMLR: Workshop and Conference Proceedings 57:120–131, 2016

Learning Deterministic Finite Automata from Infinite Alphabets

Gaetano Pellegrino G.PELLEGRINO@TUDELFT.NL
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD, Delft, The Netherlands

Christian Hammerschmidt CHRISTIAN.HAMMERSCHMIDT@UNI.LU
Interdisciplinary Institute for Security, Reliability, and Trust
University of Luxembourg
Rue Alphonse Weicker 4, L-2127, Luxembourg

Qin Lin Q.LIN@TUDELFT.NL
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD, Delft, The Netherlands

Sicco Verwer S.E.VERWER@TUDELFT.NL
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD, Delft, The Netherlands

Abstract

We proposes an algorithm to learn automata infinite alphabets, or at least too large to enumerate. We apply it to define a generic model intended for regression, with transitions constrained by intervals over the alphabet. The algorithm is based on the Red & Blue framework for learning from an input sample. We show two small case studies where the alphabets are respectively the natural and real numbers, and show how nice properties of automata models like interpretability and graphical representation transfer to regression where typical models are hard to interpret.

Keywords: Passive Learning, Deterministic Finite Automata, Regression

1. Introduction

Automata have been studied in depth, e.g. in Hopcroft et al. (2000), and successfully applied in a wide range of fields, ranging from biology over linguistics to computer science itself. Especially in the field of software specification and verification, these models are appreciated for their balance between expressive power and decidability of model and language class properties. In reverse, inferring an automaton from observations of software interactions, is an important step when analyzing and reverse engineering software and protocols, e.g. Heule and Verwer (2010), Cho et al. (2010). Here, the benefit of automata are easy to interpret and graphically representable models. Because most work on learning these models has been done under the assumption of small alphabet sizes, it is hard to transfer the nice properties to situations where the alphabet is large, or even uncountable.

In this work, our goal is to provide a general method to infer automaton models on infinite totally ordered alphabets, meant for the task of regression. Many regression methods, like ARIMA (Wei, 1994) and neural LSTM models (Gers, Schmidhuber, and Cummins, 2000), don't offer a good way of interpreting and understanding the learned model. We propose a *bottom-up approach* to learn finite automata from infinite alphabets and apply to deterministic finite automata intended for regression: each transition is constrained by intervals over the infinite alphabet. For this reason we call then Deterministic Regression Automata with Guards (RAGs).

Finite automata with potentially infinite alphabets have been studied in theory, e.g. in Neven, Schwentick, and Vianu (2004), Segoufin (2006), and shown to be conservative extensions. Often, automata are also extended with additional memory such as registers or variables Howar et al. (2012), Grumberg, Kupferman, and Sheinvald (2010). Learning these variants of automata from input samples has not been researched frequently. In the context of alphabet refinement, Howar, Steffen, and Merten (2011a) permits infinite alphabets and uses an active learning approach to learn automata. Maler and Mens (2014) proposes a modified Angluin's L* algorithm to learn from large alphabets. The method is *top-down*: Initially, the largest possible label is taken for each transition. Upon queries to the oracle, a label can be partitioned. Partitions of the infinite alphabet are used to label the transitions in the final automaton. For state-merging learning approaches, Schmidt and Kramer (2014), Schmidt, Ansorge, and Kramer (2012), propose a clustering-based algorithm to infer real-time automata on multivariate timed events, where the events may contain real-valued components. The data is used to cluster states globally in a merge step, without evaluating the equivalence of future continuations of the states.

A very close related work is Lin et al. (2016), where regression automata without guards are learned for predicting wind speed data, overcoming the limitless of the alphabet by translating it in a bounded symbolic domain.

The rest of the paper is organized as follows. In Section 2 we provide basic notation and definitions, furthermore we introduce Regression Automata with Guards. In Section 3 we describe our algorithm for learning RAGs from a data sample, and we show how it works in two case studies. In Section 4 we conclude by a discussion about the current and future work.

2. Deterministic Regression Automata with Guards

This section uses basic notation from grammar inference theory, for an introduction we refer to de la Higuera (2010). In several contexts, i.e. time series forecasting, data are sequences of points made over a continuous time interval, out of consecutive and equally spaced measurements. We model such data with sequences of symbols taken from a given totally ordered alphabet Σ, where the time is indirectly represented by the position within the sequence. This is sufficient because in practice we always deal with a finite precision of time intervals, e.g. milliseconds, minutes, hours. In this paper we introduce a new type of finite state automaton exclusively meant for dealing with large or infinite alphabets. In Regression Automata with Guards (RAGs), every transition is decorated with constraint guards. We represent a constraint guard by a closed interval in Σ, and we say that $[l, r]$ is satisfied by a symbol $s \in \Sigma$ if $s \in [l, r]$. A RAG is defined as follows:

Definition 1 (RAG) *A Regression Automaton with Guards (RAG) is a 5-tuple $\langle \Sigma, Q, q_0, \Delta, P \rangle$ where Σ is the alphabet, Q is a finite set of states, $q_0 \in Q$ is the start state, Δ is a finite set of transitions, and $P : Q \to \Sigma$ is a prediction function that assigns a prediction value to every state in Q. A transition $\delta \in \Delta$ is a triple $\langle q, q', [l, r] \rangle$ where $q, q' \in Q$ are respectively the source and target states, and $[l, r]$, $l, r \in \Sigma$, is a guard.*

We will also use functional notation for transitions, so for $q \in Q$ and $s \in \Sigma$, $\delta(q, s) = q' \in Q$ iff $\exists \langle q, q', [l, r] \rangle \in \Delta$ s.t. $s \in [l, r]$. We only focus on deterministic regression automata because of the complexity of learning non-deterministic automata (de la Higuera, 2005). A RAG is called deterministic if it does not contain two transitions with the same source state and any overlap between the guards. In a RAG, a state transition is possible only if its constraint guard is satisfied by a coming value. Hence a transition $\delta = \langle q, q', [l, r] \rangle$ is interpreted as follows: whenever the automaton is in state q, by reading an incoming value v such that $v \in [l, r]$, then the automaton changes state moving to q'.

In order to define a computation of a RAG, we introduce the notion of closest transition:

Definition 2 (closest transition) *The closest transition for a given state $q \in Q$ of a RAG $A = \langle \Sigma, Q, q_0, \Delta, P \rangle$, and given a symbol $s \in \Sigma$, is the transition $\pi_q(s)$ such that:*

$$\pi_q(s) = \begin{cases} \langle q, q_0, [s, s] \rangle & \textit{if } \nexists \langle q, q', [l, r] \rangle \in \Delta \\ \langle q, q', [l, r] \rangle & \textit{if } \exists \langle q, q', [l, r] \rangle \in \Delta \textit{ s.t. } s \in [l, r] \\ \langle q, q', [l, r] \rangle & \textit{if } \exists! \langle q, q', [l, r] \rangle \in \Delta \textit{ s.t. } s < l \textit{ or } s > r \\ \underset{\delta_{left}, \delta_{right} \in \Delta}{\operatorname{argmin}} \{ |s - r_{left}|, |s - l_{right}| \} & \textit{otherwise.} \end{cases}$$

The first branch defines a default transition to the start state when no transitions are available in q. The second branch defines the closest transition as the only one, if exists, that contains s. Since we are restricting to deterministic RAGs, at most one transition which includes the value can be present in Δ. The third branch addresses the special case when there exists only one transition in Δ, with initial state q, and it does not contain s. The last branch occurs when the value is located in between two consecutive transitions ($\delta_{left} = \left\langle q, q'_{left}, [l_{left}, r_{left}] \right\rangle$ and $\delta_{right} = \left\langle q, q'_{right}, [l_{right}, r_{right}] \right\rangle$). In this case the one with the closest edge is chosen.

The behavior of a RAG is defined by its computation:

Definition 3 (RAG computation) *A finite computation of a RAG $A = \langle \Sigma, Q, q_0, \Delta, P \rangle$ over a finite sequence of symbols $s = s_1, s_2, \ldots, s_n$ is a finite sequence*

$$q_0 \xrightarrow{s_1} q_1 \xrightarrow{s_2} q_2, \ldots, q_{n-1} \xrightarrow{s_n} q_n$$

such that for all $1 \leq i \leq n$ $\langle q_{i-1}, q'_i, [l_i, r_i] \rangle = \pi_{q_{i-1}}(s_i)$. It is also called close-computation.

Let $S \subseteq \Sigma^+$ denote a sample of non-empty sequences with symbols in Σ. We call $SUFF(S) \subseteq \Sigma^+$ the set of all non-empty suffixes of sequences in S. Hereby we introduce the notion of future continuations set of a state q, as the set of all suffixes of a sample inducing a computation in A that starts with q:

Definition 4 (future continuations set) *The future set for a state $q \in Q$ of a RAG $A = \langle \Sigma, Q, q_0, \Delta, P \rangle$, given a sample $S \subseteq \Sigma^+$, is the set $\phi_{A,S}(q) = \{s = s_1, s_2, \ldots, s_n \in SUFF(S) \mid \exists q_1, q_2, \ldots, q_n \in Q^+, q_0 = q$, and $\delta(q_{i-1}, s_i) = q_i$, $i = 1, 2, \ldots, n\}$. $\phi_A(q)$ denotes the future continuations set of state q given the sample used for learning A.*

We also introduce the notion of transition centroid given a sample as the mean of all values of the sequences, included in the sample, that get caught by this transition:

Definition 5 (transition centroid) *The centroid of a transition $\delta = \langle q, q', [l, r] \rangle$ of a RAG A, given a sample $S \subseteq \Sigma^+$, is $\mu_S(\delta) = \frac{\sum_{s \in \phi_{A,S}(q) \wedge I(s)=1} s}{\sum_{s \in \phi_{A,S}} I(s)}$, where $I(s) = \begin{cases} 1 \text{ if } |s| = 1, \\ 0 \text{ otherwise.} \end{cases}$*

When clear from the context, we will omit the sample subscript from the transition centroid.

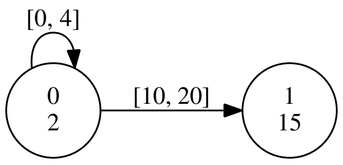

Figure 1: An example of a RAG. The leftmost state is the start state. Every state transition contains a guard. Missing transitions lead to the start state. Every state contains an identification number (above) and a prediction value (below).

Example 1 *Figure 1 shows an example of RAG. This RAG computes sequences of real values. For instance, given the sequence* 0.88, 15.07*, it crosses the states* 0, 0, 1*. Given the sequence* 0.88, 9.05 *the RAG crosses the states* 0, 0, 1 *because in state* 0 *the closest transition to value* 9.05 *leads to state* 1*.*

3. Learning Regression Automata with Guards

The problem of learning RAGs is a specialization of the more general problem of learning deterministic finite state automata (DFAs), with the additional task of learning guards over transitions. Unfortunately identifying transition guards is already an NP-Complete problem, as demonstrated in Verwer, de Weerdt, and Witteveen (2012) for time guards in real time automata. In addition, the more general problem of learning DFAs is again NP-Complete, as proved in Gold (1978). Hence we will not be able to solve this task efficiently unless P = NP. However, we can still design an efficient algorithm that learns both the structure and the guards and converge to the correct underlying RAG when more and more data are provided, in the limit. It has been done for deterministic finite state automata with RPNI algorithm (Oncina and Garcia, 1992), and for real time automata with RTI algorithm (Verwer, de Weerdt, and Witteveen, 2012). For real time automata there exists a polynomial time algorithm, able to identify time guards over transitions and structure of the automaton in the same way, such that it converges in the limit to the correct target.

Algorithm 1 Regression Automata Identifier (RAI)

```
Require: a sample $S$ of real value sequences, a threshold $\tau$
  $A :=$ BUILD-PT($S$)                                   ▷ construct the prefix tree
  $RED := \emptyset$                                         ▷ core set of red states
  $BLUE := \emptyset$                                        ▷ fringe of blue states
  PROMOTE(PROMOTE($A, RED, BLUE, q_0, \tau$))  ▷ make the root of the prefix tree red
  while $BLUE \neq \emptyset$ do
    CHOOSE($q_b \in BLUE$)                          ▷ select the best red/blue merge
    if $\exists q_r \in RED \mid$ COMPATIBLE($A, q_r, q_b, \tau$) then   ▷ if $q_b$ and $q_r$ are compatible
      MERGE($A, q_r, q_b$)                                    ▷ perform the merge
    else                              ▷ if no compatible merges are available for $q_b$
      PROMOTE($A, RED, BLUE, q_b, \tau$)                          ▷ make $q_b$ red
    end if
  end while
  return $A$
```

Algorithm 2 BUILD-PT

```
Require: a sample $S$ of real value sequences, a threshold $\tau$
  $A :=$ a RAG containing only one white start state $q_0$
  for all $s = s_0, s_1, \ldots, s_n \in S$ do             ▷ for each sequence in the sample
    $q := q_0$
    for $i = 1, 2, \ldots, n$ do                         ▷ for each value within the sequence
      if $\nexists \delta = \langle q, q', [l, r] \rangle \in \Delta \mid s_i \in [l, r]$ or $|\mu_S(\delta) - s_i| \geq \tau$ then
        $Q := Q \cup \{q'\}$
        $\Delta := \Delta \cup \{\langle q, q', [s_i, s_i] \rangle\}$          ▷ make a transition from $q$ covering $s_i$
      else
        $\exists \delta = \langle q, q', [l_\delta, r_\delta] \rangle \mid s_i \in [l_\delta, r_\delta]$
        $\delta := \langle q, q', [\min\{l_\delta, s_i\}, \max\{r_\delta, s_i\}] \rangle$        ▷ possibly update guards of $\delta$
      end if
      $\phi_{A,S}(q) := \phi_{A,S}(q) \cup \{s\,[i:n]\}$        ▷ add i-th suffix of $s$ to the future sets of $q$
      $q := q'$
    end for
  end for
  return $A$
```

Regression Automata Identifier (RAI) has been inspired by RPNI algorithm for DFAs, with which it shares the general structure. RAI (algorithm 1) starts by building a prefix tree from a given sample of sequences of real numbers (algorithm 2), then it starts merging couple of states iteratively. A merge (algorithm 3) is made only if the candidate states are compatible for merging (algorithm 5). If no compatible merge is possible for a given state, it is promoted (algorithm 7). The algorithm has as a starting point the prefix tree, that is a regression automaton. In order to avoid non-determinism, every merge is followed by a folding operation (algorithm 4). RAI implements the so called red&blue framework (de la Higuera, 2010), thus the states of a RAG are partitioned in three sets RED, $BLUE$, and

Algorithm 3 MERGE

```
Require: a RAG $A$, states $q \in RED$ and $q' \in BLUE$
  for all $\delta = \langle q_\delta, q', [l_\delta, r_\delta] \rangle \in \Delta$ do    ▷ redirect all incoming transitions in $q'$
    $\delta := \langle q_\delta, q, [l_\delta, r_\delta] \rangle$
  end for
  $\phi_A(q) := \phi_A(q) \cup \phi_A(q')$    ▷ merge future sets
  FOLD$(A, q, q')$    ▷ fix potential non-determinism
```

Algorithm 4 FOLD

```
Require: a RAG $A$, states $q, q' \in Q$, $q'$ being root of a tree
  for all $\delta_b = \langle q', q'_b, [l_b, r_b] \rangle \in \Delta$ do    ▷ scan all outgoing transitions of $q'$
    $O := \{\langle q, q'_o, [l_o, r_o] \rangle \in \Delta \mid [l_o, r_o] \cap [l_b, r_b] \neq \emptyset\}$ ▷ set of transitions of $q'$ that overlap $\delta_b$
    if $O = \emptyset$ then    ▷ there is no overlap, thus just redirect $\delta_b$
      if $q \notin RED$ then    ▷ guards will get built later
        $\delta_b := \langle q, q'_b, [l_b, r_b] \rangle$
      else    ▷ $q \in RED$, hence guards have already been built
        $closest := \pi_q(\mu(\delta_b))$
        JOIN-TRANSITIONS$(clostest, \delta_b)$
      end if
    else
      for all $\delta_o = \langle q, q'_o, [l_o, r_o] \rangle \in O$ do
        FOLD$(A, q'_o, q'_b)$    ▷ otherwise recursively fold all subtrees
      end for
    end if
  end for
```

WHITE. One invariant of RAI is that *RED* is a core of states representing the already identified part of the target automaton structure, surrounded by a fringe of blue states, and remaining states are white. Only merges between red and blue states are allowed in RAI. Blue states have only one predecessor, and they are root of a tree.

Algorithm 3 takes a red state q and a blue state q'. It first redirects all incoming transitions of q' to q. Then it merges the future set of q' with the future set of q. After that, the tree rooted in q' is disconnected, thus it is folded in the rest of the RAG. Algorithm 4 is in charge of that.

Algorithm 5 scans every future continuation f_r of the red state candidate for merging. Then it pair f_r with the closest future continuation of the blue candidate, according to the average prefix euclidean distance (1). It does the same for every future continuation f_b of the blue candidate. If the average distance among continuations is lower than a given threshold, then it returns YES.

$$p(f, f') = \frac{\sqrt[2]{\sum_{i=1}^{\min\{|f|,|f'|\}} (f_i - f'_i)^2}}{\min\{|f|, |f'|\}} \tag{1}$$

The CHOOSE function in RAI just selects the red/blue couple that minimizes the score computed by algorithm 6. Basically it is similar to COMPATIBLE, but here it just sums

Algorithm 5 COMPATIBLE

```
Require: states $q \in RED$ and $q' \in BLUE$, a threshold $\tau$
  $sum := 0$
  for all $f_r \in \phi_A(q)$ do
    $f_b :=$ CLOSEST-FUTURE($q', f_r$)
    $sum := sum + p(f_r, f_b)$          ▷ $p(\cdot,\cdot)$ is the average prefix euclidean distance
  end for
  for all $f_b \in \phi_A(q')$ do
    $f_r :=$ CLOSEST-FUTURE($q, f_b$)
    $sum := sum + p(f_b, f_r)$          ▷ $p(\cdot,\cdot)$ is the average prefix euclidean distance
  end for
  return $sum/ (|\phi_A(q)| \times |\phi_A(q')|) < \tau$
```

Algorithm 6 SCORE

```
Require: states $q \in RED$ and $q' \in BLUE$
  $r := 0$
  for all $f_r \in \phi_A(q)$ do
    $f_b :=$ CLOSEST-FUTURE($q', f_r$)
    $r := r + p(f_r, f_b)$          ▷ $p(\cdot,\cdot)$ is the average prefix euclidean distance
  end for
  return $r$
```

out all the distances between red and blue futures. The idea behind such a criterion is to choose couples of states that show closeness in all the sequences observed in the training sample that originates form both.

Algorithm 7 reacts differently depending on the partition the input state q belongs to. If $q \in RED$ no further promotion is possible, thus the algorithm does nothing. If $q \in BLUE$ then it is moved to the red partition and all its successors, white by definition, get promoted to $BLUE$. If $q \in WHITE$ it is moved to $BLUE$, and then the time guards for the outgoing transitions are identified.

Algorithm 8 identifies guards by grouping transitions which are close each other. It scans all outgoing transitions of a given state q in ascending order of the unique value each of them represents. We are certain that all such transitions represent only one value because q is root of a subtree of the prefix tree built at first step of RAI. Then it decides whether to join a given transition to the previous one or not. The decision is made according to the average prefix euclidean distance among all possible pairs of sequences in the future sets passing through both transitions.

Figure 2 shows how RAI works on the sample $S = \{\langle 1,2,1\rangle, \langle 1,13\rangle, \langle 2,2,2,14\rangle, \langle 2,13\rangle, \langle 17\rangle\}$ in order to identify the model of Figure 1. It firstly builds the prefix tree (a) by calling Algorithm 2, then it chooses states 0 and 1 for merging. Notice that 0, the root of the tree, has been previously promoted to RED thus 1 must have been promoted to $BLUE$ as well. After the merge (b) Algorithm 4 is called on the same couple of states, thus transition to 2 is added to the cluster represented by the self loop in 0, and also transitions to 6 and 4 get clustered together (c). Algorithm 4 calls itself recursively on states 0 and 2, thus transition

Algorithm 7 PROMOTE

```
Require: a RAG $A$, sets $RED$, $BLUE \subseteq Q$, a state $q \in Q$, a threshold $\tau$
  if $q \in RED$ then                                              ▷ $q$ is red
      return $q$                        ▷ a red state cannot be promoted any further
  end if
  if $q \in BLUE$ then                                             ▷ $q$ is blue
      $BLUE := BLUE \setminus \{q\}$
      $RED := RED \cup \{q\}$
      for all $\langle q, q', [l, r]\rangle \in \Delta$ do      ▷ for all outgoing transition of $q$
          if $q' \notin BLUE$ and $q' \notin RED$ then                ▷ if $q'$ is white
              PROMOTE($A, RED, BLUE, q'$)
          end if
      end for
      BUILD-GUARDS($A, q, \tau$)
      return $q$
  end if
  $BLUE := BLUE \cup \{q\}$                                        ▷ $q$ is white
  return q
```

Algorithm 8 BUILD-GUARDS

```
Require: a RAG $A$, a state $q$ being root of a tree, a threshold $\tau$
  sort singleton transitions of $q$ in ascending order of their unique value
  $last := \epsilon$                            ▷ $last$ is the last encountered transition
  for all $\delta = \langle q, q', [l, r]\rangle \in \Delta$ do     ▷ for all outgoing sorted transitions of $q$
      if $last = \epsilon$ or $|\mu(last) - \mu(\delta)| \geq \tau$ then  ▷ $\delta$ cannot be joined to the $last$ transition
          $last := \delta$
      else                              ▷ join $\delta$ to the $last$ encountered transition
          JOIN-TRANSITIONS($\delta, last$)
      end if
  end for
```

from 0 to 3 in model (c) get added to the self loop in 0 (d). After an additional recursive call the folding process ends in model (e), whose structure is the same of the target.

Figure 3 shows how RAI works on a sample of 10000 observations generated according to a sinus wave, thus a potentially infinite alphabet $\mathbb{R}$. The threshold τ, used for the compatibility check, has been set to 0.3. RAI minimizes a prefix tree of 17848 states, into a RA of just three. It is possible to interpret such states, by looking at the outgoing transitions from each of them. State 0, for instance, explains the first 180^o of the sinus period.

4. Discussion

We have defined a generic algorithm for learning regression automata, targeting languages with large or infinite alphabets that can be recognized by models with limited number of states and transitions. There exist some works which address the same issue, mainly in the active learning domain where models are learned by querying informed oracles (Maler and

Algorithm 9 JOIN-TRANSITIONS

```
Require: a RAG $A$, a transition $\delta_c = \langle q_c, q'_c, [l_c, r_c]\rangle$ to join to transition $\delta_j = \left\langle q_j, q'_j, [l_j, r_j]\right\rangle$
  $\delta_j := \left\langle q_j, q'_j, [\min\{l_o, l_j\}, \max\{r_o, r_j\}]\right\rangle$        ▷ possibly update guards of $\delta_j$
  $\Delta := \Delta \setminus \{\delta_c\}$        ▷ drop $\delta_c$ transition from $A$
  $\phi_A(q'_j) := \phi_A(q'_j) \cup \phi_A(q'_c)$        ▷ update future continuations of $q'_j$
  for all $\delta_o = \langle q_c, q'_o\, [l_o, r_o]\rangle \in \Delta$ do        ▷ update outgoing transitions of $q_c$
      $\delta_{ov} := \langle q_{ov}, q'_{ov}, [l_{ov}, r_{ov}]\rangle \in \Delta \mid l_{ov} \leq \mu(\delta_o) \leq r_{ov}$        ▷ $\delta_{ov}$, if exists, overlaps $\delta_o$
      if $\delta_{ov} = \epsilon$ then
          $\delta_o := \langle q_j, q'_o\, [l_o, r_o]\rangle$
      else
          JOIN-TRANSITIONS($\delta_{ov}, \delta_o$)
      end if
  end for
```

Mens 2014, Veanes et al. 2012). However they assume a radically different approach because they aim to partition the alphabet in a top-down strategy. Also Verwer, de Weerdt, and Witteveen (2012), in RTI, uses the same approach to split guarded transitions before merging couples of states. Our algorithm follows a bottom-up approach, identifying transitions by merging consecutive transitions instead of splitting them.

The genericity of the algorithm comes from the adoption of the Red and Blue framework for learning automata from samples (de la Higuera, 2010), with the addition of a grouping procedure for identifying non-overlapping subsets of the alphabet in each state. Of course each problem requires its own specific merging criterion between states, as well as its own grouping strategy. In this work we have presented an instantiation of this algorithm for both alphabets $\Sigma = (\mathbb{R}, \leq)$ and $\Sigma = (\mathbb{N}, \leq)$. When dealing with numbers, the grouping into a finite number of intervals is very natural and may be used in many application domains for regression and prediction. In the next future we are planning to compare the predictive power of regression automata with other approaches for numerical time series, and to compare our algorithm with other inference techniques meant for sequences of numbers.

Acknowledgments

This work was partially supported by Technologiestichting STW VENI project 13136 (MANTA) and (LEMMA), as well as FNR Luxembourg AFR project PAULINE.

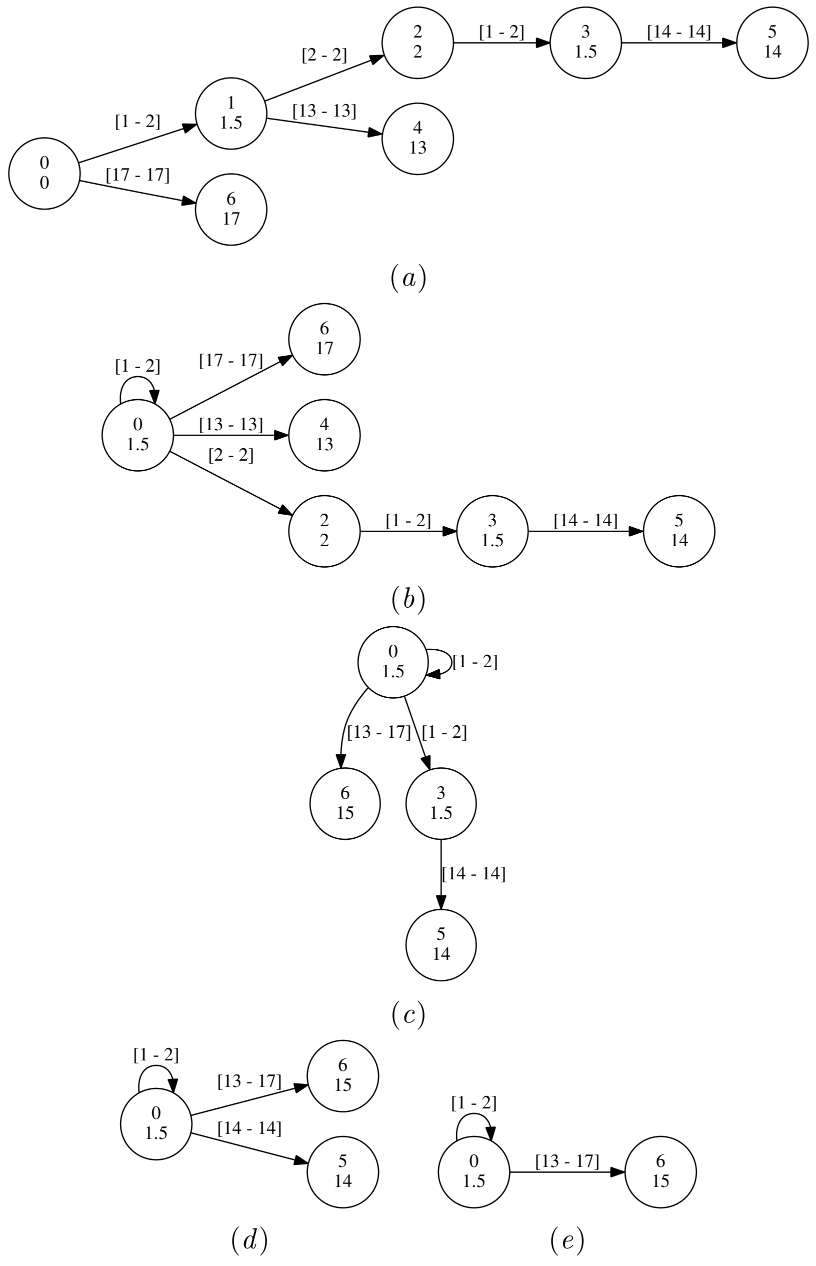

Figure 2: RAI algorithm in action to learn the model in Figure 1 from the sample $S = \{\langle 1, 2, 1\rangle, \langle 1, 13\rangle, \langle 2, 2, 2, 14\rangle, \langle 2, 13\rangle, \langle 17\rangle\}$ and threshold $\tau = 5$. (a): prefix tree construction. (b): after merging states 0 and 1. (c): after folding subtree rooted in 1 in subtree rooted in 0. (d): after folding subtree rooted in 2 in subtree rooted in 0. (d): after folding subtree rooted in 3 in subtree rooted in 0.

References

Chia Yuan Cho, Eui Chul Richard Shin, Dawn Song, et al. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th*

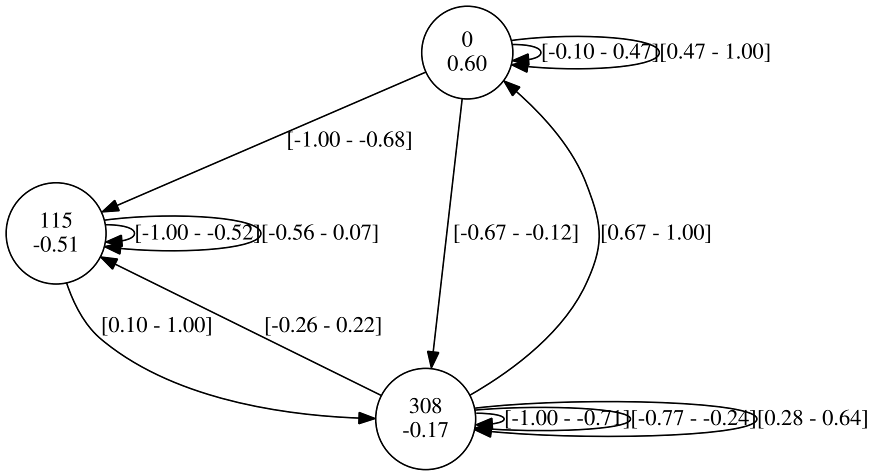

Figure 3: Regression Automaton learned by RAI algorithm from a sample of 10000 sequences generated by sinus wave. The prefix tree contains 17848 states. Threshold (τ) has been set to 0.3. State 0 covers the first 180^o, state 115 the second 180^o. State 308 is a transition state between the two halves of a sinus period.

ACM conference on Computer and communications security, pages 426–439. ACM, 2010.

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

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

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

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

Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. Variable automata over infinite alphabets. In *Language and Automata Theory and Applications*, pages 561–572. Springer, 2010.

Marijn JH Heule and Sicco Verwer. Exact dfa identification using sat solvers. In *Grammatical Inference: Theoretical Results and Applications*, pages 66–79. Springer, 2010.

John E. Hopcroft, Rajeev Motwani, Rotwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computability.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000.

Falk Howar, Bernhard Steffen, and Maik Merten. Automata learning with automated alphabet abstraction refinement. In *Verification, Model Checking, and Abstract Interpretation*, pages 263–277. Springer, 2011a.

Falk Howar, Bernhard Steffen, and Maik Merten. Automata learning with automated alphabet abstraction refinement. In *Verification, Model Checking, and Abstract Interpretation*, pages 263–277. Springer Berlin Heidelberg, 2011b.

Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring canonical register automata. In *Verification, Model Checking, and Abstract Interpretation*, pages 251–266. Springer, 2012.

Qin Lin, Christian Hammerschmidt, Gaetano Pellegrino, and Sicco Verwer. Short-term time series forecasting with regression automata. In *SIGKDD 2016 Workshop on Mining and Learning from Time Series (MiLeTS)*, 2016.

Oded Maler and Irini-Eleftheria Mens. Learning regular languages over large alphabets. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 485–499. Springer, 2014.

Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. 5(3):403–435, 2004.

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

Jana Schmidt and Stefan Kramer. Online induction of probabilistic real-time automata. 29 (3):345–360, 2014.

Jana Schmidt, Sonja Ansorge, and Stefan Kramer. Scalable induction of probabilistic real-time automata using maximum frequent pattern based clustering. In *SDM*, pages 272–283. SIAM, 2012.

Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Computer Science Logic*, pages 41–57. Springer, 2006.

Margus Veanes, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaj Bjorner. Symbolic finite state transducers: Algorithms and applications. 47(1):137–150, 2012.

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

William Wu-Shyong Wei. *Time series analysis*. Addison-Wesley publ Reading, 1994.