Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Learning Behavior Models for Hybrid Timed Systems

Oliver Niggemann
inIT – Institute Industrial IT,
Fraunhofer IOSB – Competence Center Industrial Automation
32657 Lemgo, Germany
oliver.niggemann@iosb-ina.fraunhofer.de

Benno Stein
Bauhaus-Universität Weimar
99423 Weimar, Germany
benno.stein@uni-weimar.de

Alexander Maier
inIT – Institute Industrial IT
32657 Lemgo, Germany
alexander.maier@hs-owl.de

Asmir Vodenčarević and **Hans Kleine Büning**
University of Paderborn
33098 Paderborn, Germany
{asmir.vodencarevic, kbcsl}@upb.de

Abstract

A tailored model of a system is the prerequisite for various analysis tasks, such as anomaly detection, fault identification, or quality assurance. This paper deals with the algorithmic learning of a system's behavior model given a sample of observations. In particular, we consider real-world production plants where the learned model must capture timing behavior, dependencies between system variables, as well as mode switches—in short: hybrid system's characteristics. Usually, such model formation tasks are solved by human engineers, entailing the well-known bunch of problems including knowledge acquisition, development cost, or lack of experience.

Our contributions to the outlined field are as follows. (1) We present a taxonomy of learning problems related to model formation tasks. As a result, an important open learning problem for the domain of production system is identified: The learning of hybrid timed automata. (2) For this class of models, the learning algorithm HyBUTLA is presented. This algorithm is the first of its kind to solve the underlying model formation problem at scalable precision. (3) We present two case studies that illustrate the usability of this approach in realistic settings. (4) We give a proof for the learning and runtime properties of HyBUTLA.

Keywords: Model Formation, Simulation, Machine Learning, Technical Systems

Introduction

The manual creation of models by human experts is the main bottleneck during the development of algorithms for model-based anomaly detection or model-based diagnosis. The learning of task-specific models, more precisely, *the automation of model formation by machine learning*, is a promising approach to overcome this modeling bottleneck. However, while the learning of simple model types can be considered as being state-of-the-art, model learning is still a challenge for many relevant system classes. This is especially true for behavior models of those technical systems where intricate aspects such as timing, mode changes, or hybrid signals are to be captured.

In this paper, we first identify hybrid timed automata as a crucial class of models for technical systems such as production plants. For this identification, a new taxonomy of models (and corresponding learning algorithms) is defined in this section. Because the learning of such models is an open research issue, the paper then presents a new suitable learning algorithm: HyBUTLA. This algorithm is studied empirically using two case studies and a theoretical analysis.

The learning of behavior models is a complex problem and touches several research fields: machine learning, systems engineering, systems analysis and simulation. To organize the different problems and their complexities we have developed a classification scheme, organized in the Tables 1 and 2 below. The scheme is oriented at two widely-used model classification principles (dimensions) in engineering and systems theory (Wymore 1976; Zeigler, Praehofer, and Kim 2000; Stein 2001): (1) the different forms of a model's state space (the x-axis in the tables), and (2) the model's structural variability of time as expressed by its number of modes (the y-axis in the tables). In addition, we will explicitly distinguish models whose behavior is governed by stochastic elements.

Our scheme is intended to be complete, in order to be useful as a landscape of the entire spectrum of problems that one may encounter in practice. In this regard Table 1 organizes the model formation problems, i.e., the systems engineering view, while Table 2 organizes the respective machine learning perspective. With respect to the state space dimension (x-axis) the following five model types are distinguished:

Memoryless Models. Memoryless models cannot undergo a state change; their output depend in a definite way on their input. Models of this type are also called stationary models.

Dynamic Models (with finite state number). The system behavior is comprised of a sequence of signal values where time is modeled explicitly. Each new model state is com-

Table 1: A classification of models oriented at the dimensions *model dynamics* and *model function*.

Complexity of model function	Memoryless models	Dynamic models (finite states)	Dynamic models (infinite states)	Dynamic stochastic models (finite states)	Dynamic stochastic models (infinite states)
Multi-mode	∅	Discrete models Example: hierarchical automaton	Hybrid models Example: hydraulic circuit	Discrete models with random states Example: production system	Hybrid models with random states Example: process industry
Single-mode	Function models Example: resistor	Discrete models Example: temporal automaton	Differential-algebraic models Example: oscilator circuit	Discrete models with random states Example: robot with defects	Differential-algebraic models with random states Example: defect transitor

Complexity of model dynamics (low ↔ high)

puted based on the state's predecessors. Examples include temporal automata and temporal logic.

Dynamic Models (with infinite state number). Given a constant input signal and some internal arbitrary state, the behavior over time is created by iteratively computing all signal values along with their derivatives. I.e., the time characteristic is computed implicitly by a numerical solver.

Dynamic Stochastic Models (with finite state number). Dynamic stochastic models show a non-deterministic behavior. This may be caused by random effects such as errors or by asynchronous parts of the systems that run independently and show therefore a stochastic timing behavior.

Dynamic Stochastic Models (with infinite state number). These models are similar to the class of "dynamic stochastic models with finite state number" but comprise an infinite number of states.

With respect to the structural variability (y-axis) the following differentiation is made:

1. *Single-Mode Models.* These models show a behavior that can be defined by a function from C^0 or C^1, i.e., a continuous function or a differentiable function whose derivative is continuous.
2. *Multi-Mode Models.* These models are designed to capture a variety of functionalities. For each such functionality a so-called mode is provided. Basically, every system that can fundamentally change its behavior, such as a vehicle that changes its gear or a valve in a chemical reactor that is opened, shows multi-mode behavior (Buede 2009).

Apart from the above classification our contributions are centered around the box shown gray in Table 2, the learning hybrid, timed automata. A new algorithm HyBUTLA is explained, which is the first learning algorithm for hybrid timed automata. HyBUTLA is here studied by means of two use cases and by a theoretical analysis.

The remainder of the paper is organized as follows. Oriented at the increasing complexity of the learning problem we first overview related work. We then present the case studies, and finally report on our theoretical findings.

Table 2: On the *learnability* of model behavior, considering the model types in Table 1.

Complexity of model function	Memoryless models	Dynamic models (finite states)	Dynamic models (infinite states)	Dynamic stochastic models (finite states)	Dynamic stochastic models (infinite states)
Multi-mode	∅	Rule learning Examples: (Verwer, de Weerdt, and Witteveen 2010)	Hybrid Timed Automata Learning	Learnable?	Learnable?
Single-mode	Function approximation Example: Regression	Rule learning Examples: (Verwer, de Weerdt, and Witteveen 2010)	Learning of DAEs Example: (Schmidt and Lipson 2009)	Learnable?	Learnable?

Complexity of model dynamics (low ↔ high)

Related Work

Single-mode, memoryless models can be learned using function approximation approaches: for this, a function template is given a-priori and is then parameterized by minimizing the prediction error of the function. Generally speaking such approaches are well studied (Markou and Singh 2003a; 2003b). Single-mode, dynamic models are harder to learn because the states have to be identified. For a finite number of states, this can be done by learning (temporal) rules (Witten and Frank 2005) or by learning of (timed) automata:

For the learning of automata several algorithms exist, e.g. MDI (Thollard, Dupont, and de la Higuera 2000) and ALERGIA (Carrasco and Oncina 1999) for the learning from positive samples. Learning timed automata is a rather new field of research, e.g. in (Maier et al. 2011; Verwer, de Weerdt, and Witteveen 2010).

For single-mode dynamic models with an infinite number of states, the problem is even harder: Only a few projects try to identify such equation systems (Schmidt and Lipson 2009). As a work-around, often only parameters of an a-priori model are identified by means of optimization algorithms (Isermann 2004).

No established learning algorithms exist for multi-mode timed models, so the algorithmic gap opens exactly for those models suitable for technical systems. This paper tries to close this gap.

In general, no learning algorithms exist for stochastic dynamic models. E.g. it is not possible to identify asynchronous system parts based on observations of the overall system.

Model Learning Algorithm

In this section, the new algorithm HyBUTLA for the learning of hybrid timed automata is given. As outlined before, such learning algorithms are currently a focus point of machine learning—especially for technical systems. Later on, two use cases for this algorithm are described.

Figure 1 shows an example of a hybrid timed automaton.

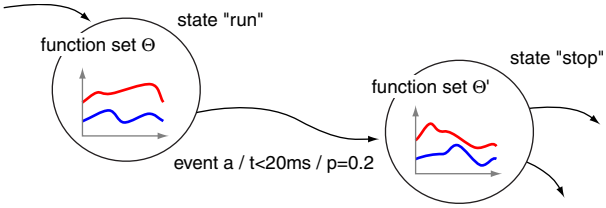

Figure 1: Example of a hybrid automaton.

A transition between two states *run* and *stop* is triggered by an event a—this often corresponds to a mode change of the system. The transition is only taken if a timing constraint $t < 20ms$ holds. Additionally, a probability $p = 0.2$ for taking the transition is given. Within a state, continuous signals may change according a function θ.

In the following, a hybrid timed automaton is defined:

Definition 1. *(Hybrid Timed Automaton) A hybrid timed automaton is a tuple* $A = (S, s_0, F, \Sigma, T, \Delta, Num, c, \Theta)$*, where*

- S *is a finite set of states,* $s_0 \in S$ *is the initial state, and* $F \subseteq S$ *is a set of final states,*
- Σ *is the alphabet comprising all relevant events.*
- $T \subseteq S \times \Sigma \times S$ *gives the set of transitions. E.g. for a transition* $\langle s, a, s' \rangle$*,* $s, s' \in S$ *are the source and destination states and* $a \in \Sigma$ *is the trigger event.*
- *A set of transition timing constraints* Δ *with* $\delta : T \rightarrow I, \delta \in \Delta$*, where* I *is the set of time intervals.*
- *A function* $Num : T \rightarrow \mathbf{N}$ *counts the number of observations that used the transition.*
- *A single clock* c *is used to record the time evolution. At each transition, the clock is reset. This allows only for the modeling of relative time steps.*
- *A set of functions* Θ *with elements* $\theta_s : \mathbf{R}^n \rightarrow \mathbf{R}^m, \forall s \in S, n, m \in \mathbf{N}$*. I.e.* θ_s *is the function computing signal value changes within a single state* s*.*

Please note that the function Num can be used to compute transition probabilities p for a transition $\langle v, a, w \rangle \in T$: $p(v, a, w) = \frac{Num(v,a,w)}{\sum_{(v,b,w') \in T, b \in \Sigma, w' \in S} Num(v,b,w')}$.

The manual creation of such a hybrid timed automaton is often not possible, e.g. because experts are not available. Therefore in (Vodenčarević et al. 2011; Maier et al. 2011), first ideas for corresponding model learning algorithms were given. The algorithm is defined formally in algorithm 1 and also sketched in figure 2.

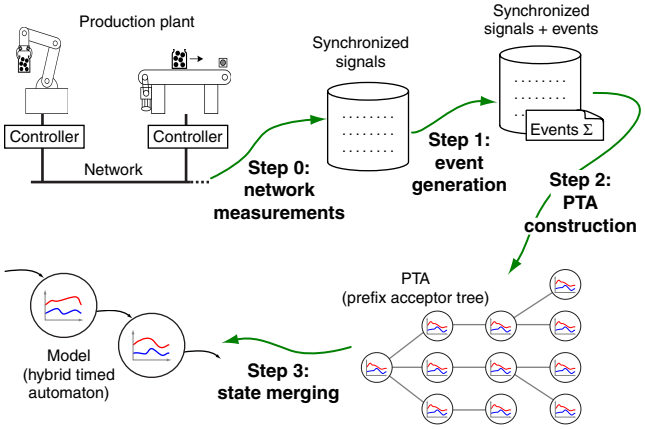

Figure 2: General concept for learning hybrid timed automata, numbers refer to the step of the algorithm 1.

The algorithm works as follows: First in **step (0)**, all relevant signals are measured during a system's normal operation and stored in a database. This measurement approach for distributed plants is described in (Pethig and Niggemann 2012).

In **step (1)**, events are generated. An event is generated whenever a discrete variable changes its value—this often

corresponds to an actuator or sensor signal in a technical system such as switching a valve or the toggling of a photoelectric barrier. These signals are used later on to trigger transitions between states, i.e. these events define mode switches. Furthermore, more events may be generated whenever a continuous signal crosses a, manually defined, threshold.

Often, some events (e.g. turning on an actuator) appear several times in the set of observations. In this case it must be checked whether these events are generated by the same process; this is done based on the events' timing. As described in definition 1, an event's timing is defined as the time span since the last event occurrence. So for each available event a, the Probability Density Function (PDF)—probability over time—is computed. If the PDF is the sum of several Gaussian distributions, separate events are created for each Gaussian distribution.

E.g. in figure 3, the signal a controlling the robot is used for two different processes with two different timings (i.e. PDFs): Containers are sorted according to their size, each size results in a different timing of the robot movements.

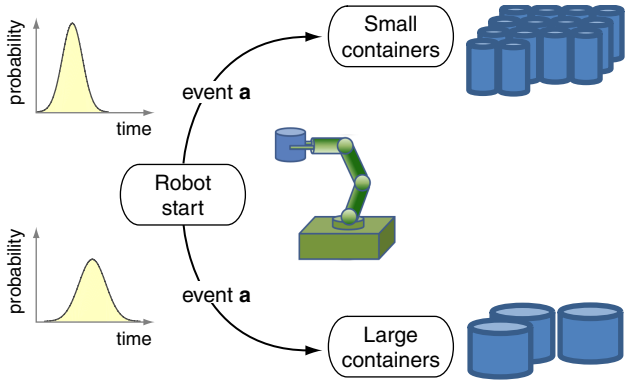

Figure 3: An event which is used in different process contexts is treated as two different events.

In **step (2)**, a prefix tree acceptor (PTA) is built. Such a PTA is a hybrid timed automaton in the form of a tree. In a PTA, each sequence of observations results in one path from the tree's root to a leaf; common prefixes of sequences are therefore stored only once.

The continuous behavior (i.e. the output process variables) within each state $s \in S$ is modeled in **step (2.1)** by learning the functions $\theta_s \in \Theta$ from definition 1. To this aim, machine learning methods such as linear regression or neural networks are applied using the portion of recorded sensor measurements associated with the corresponding state: Whenever possible, a linear regression is used. If the linear regression leads to a too high approximation error, a neural network with the following structure has been used: feed-forward network, 1 hidden layer, 25 neurons, tan-sigmoid transfer function, output layer with linear transfer function, Levenberg-Marquardt back-propagation.

Now in **step (3)**, the compatible states are merged until a smaller automata is reached, which can still predict the system behavior: First, in **step (3.1)**, the compatibility between two states is checked in the bottom-up order. The bottom-up order of the nodes is defined here by the lexicographic order of the shortest sequence of events leading to the node.

If the states are found to be compatible, they are merged in **step (3.1.1)**. The compatibility criterion consists of several similarity tests for the two nodes and is also applied recursively to the nodes' subtrees. This compatibility criterion will be described later. When two states are merged, their portions of observed data are combined and new functions θ are learned. Because after the merging step the resulting automaton may be non-deterministic, the subtrees of the new merged state are made deterministic in step **(3.1.2)** by merging their nodes recursively (see also (Carrasco and Oncina 1999) for details).

Whenever two nodes are merged, the transition probabilities (here defined by the function Num from definition 1) must be recalculated. In general, also the functions $\theta_s \in \Theta$ from definition 1 must be relearned. But for performance reasons, this is only done once after the final automation has been identified.

Because of this recursive determinization step, the order in which the nodes are merged is crucial for the algorithm's runtime performance. The new bottom-up merging order used here means a significant speed-up of the algorithm compared to previous algorithms (Thollard, Dupont, and de la Higuera 2000; Carrasco and Oncina 1999; Verwer 2010). Figure 4 shows an example: On the left hand side, two nodes will be merged. Before the merging step, both subtrees must be checked for compatibility, this takes $O(n + n')$ steps. After the merging step, the merged subtree comprises $n+n'$ nodes and has to be determinized, this again takes $O(n + n')$ steps. After this merging step, further merging steps are done within the subtree, and again, further recursive compatibility checks and recursive determinization steps are required.

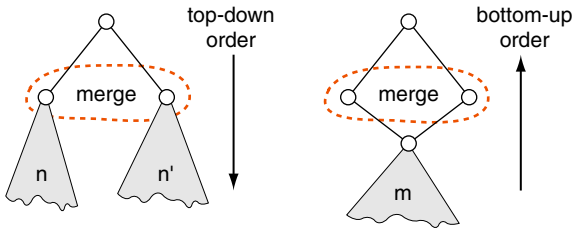

Figure 4: The advantage of a bottom-up merging order.

The example on the right hand side shows the result of a bottom-up order merging: Since the subtree has already been merged in previous merging steps, the subtrees share a large parts of their nodes and the compatibility can therefore be checked efficiently. Furthermore, no recursive determinization of the subtree is necessary because its has been deterministic before. This is a rather optimist example, but it captures the gist of the bottom-up strategy. A formal analysis can be found later in this paper.

A very important issue is the compatibility criterion for merging two states: Algorithm 2 gives a comparison

Algorithm HyBUTLA (Σ, $\mathcal{O}$):
Given:
(1) Events Σ
(2) Observations $\mathcal{O} = \{\mathbf{O}_0, \ldots, \mathbf{O}_{n-1}\}$ where $\mathbf{O}_i \in (\Sigma \times \mathbf{R})^*$, $\mathbf{O}_i$ is one sequence of timed events (e.g. a system cycle)
Result: Hybrid Automaton $\mathcal{A}$
(1) Compute events Σ based on $\mathcal{O}$.
(2) Build prefix tree $PTA = (S, s_0, F, \Sigma, T, \Delta, Num, c, \Theta)$ based on $\mathcal{O}$. Let S' be all non-leaf nodes in S.
(2.1) $\forall s \in S$ learn $\theta_s \in \Theta$ using continuous data from $\mathcal{O}$. PTA is a hybrid automaton according to definition 1.
(3) **for all** $v, w \in S'$ in a bottom-up order **do**
(3.1) **if** compatible(v, w) **then**
(3.1.1) $\mathcal{A}$ = merge(v, w)
(3.1.2) determinize($\mathcal{A}$) **od**
(4) **return** $\mathcal{A}$

Algorithm 1: Hybrid automata identification algorithm HyBUTLA.

Algorithm compatible (v,w):
Given: $v, w \in S$
Result: decision yes or no
(1) $f(v,a) := \sum_{e=(v,a,*)\in T} Num(e), v \in S, a \in \Sigma$ where $*$ is an arbitrary element
(2) $f_{in}(w) := \sum_{e=(*,*,w)\in T} Num(e), w \in S$
(3) $f_{out}(v) := \sum_{e=(v,*,*)\in T} Num(e), v \in S$
(4) $f_{end}(v) := f_{in}(v) - f_{out}(v), v \in S$
(5) **if** fractions-different($f_{in}(v), f_{end}(v), f_{in}(w), f_{end}(w)$)
(5.1) **return false**
(6) **for all** $a \in \Sigma$ **do**
(6.1) **if** fractions-different($f_{in}(v), f(v,a), f_{in}(w), f(w,a)$)
(6.1.1) **return false**
(6.2) **if** not compatible(v', v'') $\forall\ (v,a,v'), (w,a,v'') \in T$
(6.2.1) **return false od**
(7) **return true**

Algorithm 2: The algorithm for defining the compatibility of two nodes in the prefix tree.

function for testing the similarity of two states—following ALERGIA (Carrasco and Oncina 1999). The main idea is to check whether the probabilities for for stopping in the state (step 5 of algorithm 2) or for taking a specific transition (step 6.1 of algorithm 2) are rather similar for both states. This similarity is checked using the function *fractions-different*. This function evaluates the Hoeffding Bound, which compares whether two fractions $\frac{f_0}{g_0}$ and $\frac{f_1}{g_1}$ are significantly different:

$$\text{fractions-different}(g_0, f_0, g_1, f_1) := \left|\frac{f_0}{g_0} - \frac{f_1}{g_1}\right| > \sqrt{\frac{1}{2}\log\left(\frac{2}{\alpha}\right)}\left(\frac{1}{\sqrt{g_o}} + \frac{1}{\sqrt{g_1}}\right), f_0, g_0, f_1, g_1 \in \mathbf{N} \quad (1)$$

where $1-\alpha, \alpha \in \mathbf{R}, \alpha > 0$ is the probability of the decision. If this inequality is true, the probabilities are different, i.e. states are different and will not be merged.

If the two nodes are found to be compatible, the compatibility of the respective subtrees must be checked too. This is done by applying the algorithm 2 recursively to all nodes in the subtrees (step 6.2 of algorithm 2).

Learning of Hybrid Automata for Energy Anomaly Detection

The automatic detection of suboptimal energy consumptions in production plants is a key challenge for the European industry in the next years (Group 2006; VDI/VDE 2009). Currently, operators are mainly concerned about a plant's correct functioning; energy aspects are still regarded as of secondary importance. And even if energy aspects are seen as a problem, operators mainly replace single hardware component. Details about these approaches can be found e.g. in (EWOTeK ; EnHiPro).

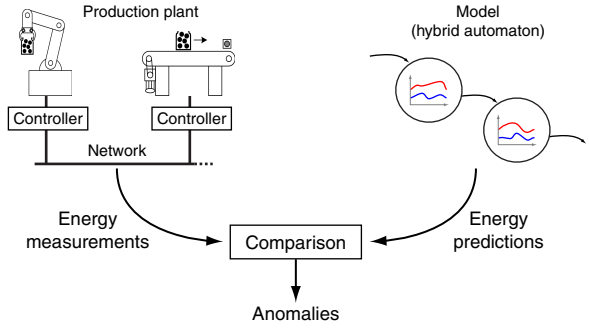

Figure 5: Detecting energy anomalies using hybrid automata.

Suboptimal energy situations can be detected by means of model-based anomaly detection as depicted in figure 5:

A model is used to simulate the normal energy consumption of a plant. For this, the simulation model, here a hybrid automaton, needs all inputs of the plant, e.g. product information, plant configuration, plant status, etc. If the actual energy measurements vary significantly from the simulation results, the energy consumption is classified as anomalous. Such an anomaly detection algorithm is given in (Vodenčarević et al. 2011).

Here we present results obtained using the Lemgo Model Factory (LMF), i.e. an exemplary production plant for bulk goods. The learned model of this production plant comprised 19 states (around 63% smaller than the original PTA). The (continuous) output variable is the machine's active power measured in a range of $0-3.250kW$. HyBUTLA uses measurements of 12 production cycles comprising 6 discrete and 6 continuous signals for the learning.

There were 30% of anomalous samples in the test cycle. The targeted anomalies in the continuous part of the system were signal zero value, signal drop by 10%, and signal jump

by 10%. Performance metrics included the sensitivity, the specificity, and the overall accuracy (Fawcett 2006). The experiment was repeated 100 times, and its summarized results are given in table 3. All anomalies in the discrete part of the system were successfully detected.

Table 3: Performance metrics for the real data.

Performance	Signal faults		
	Zero value	Drop by 10%	Jump by 10%
Sensitivity (%)	100	97.20	97.29
Specificity (%)	100	91.66	97.72
Accuracy (%)	100	93.35	97.59

In order to demonstrate the scalability of the HyBUTLA algorithm, artificial data was generated. The dataset is 5 times larger compared to the real plant (31 continuous and 30 discrete signals). The learned model had a small size, 72 states compared to 170 states in the PTA. Again 30% of the samples were anomalous and the experiment was repeated 100 times. Table 4 shows the performance metric. These results demonstrate the applicability of learned models for anomaly detection in larger systems.

Table 4: Performance metrics for the artificial data.

Performance	Signal faults		
	Zero value	Drop by 10%	Jump by 10%
Sensitivity (%)	100	82.83	83.11
Specificity (%)	100	97.58	96.32
Accuracy (%)	100	93.15	92.35

Learning of Timed Automata

Learning the timing behavior of discrete manufacturing plants is currently an important challenge, e.g. in (Preuße and Hanisch 2009). Such timing models can be used to analyze a plant's timing behavior and are the basis for tasks such as system optimization or anomaly detection.

During the anomaly detection phase, the running plant's timing behavior is compared to the simulation results as outlined in figure 5. A timing anomaly is signaled whenever a measured timing is outside the timing interval δ (definition 1) in the learned hybrid timed automaton. Here, the interval is defined as $[\mu - k \cdot \sigma, \mu + k \cdot \sigma], k \in \mathbf{R}_+$ where μ is the mean value of the corresponding original observations' timings and σ is the standard deviation.

In a first experiment, the Lemgo Model Factory is used again. A frequently occurring error for example is the wear of a conveyer belt which leads to a decrease in the system's throughput. 12 production cycles are used to learn a normal behavior model. The PTA comprises 6221 states. HyBUTLA reduces this to 13 states—this corresponds to a compression rate of 99.79%.

To verify the model learning algorithm with a high amount of data, in a second experiment, data is generated artificially using the Reber grammar (see (Reber 1967) for details) and additionally extended with timing information. 1000 samples are generated to learn the model, then 2000 test samples are created where 1000 comprise timing errors. From the initial 5377 states in the PTA, a model with 6 states is learned.

Table 5 shows the error rates for the anomaly detection applied to both data sets using different factors k.

Table 5: Experimental results using real and artificial data.

	k =1	k =2	k =3	k =4
error rate (%) - LMF	2	5.3	12.8	30
error rate (%) - Reber grammar	0	1.3	7.5	21

Both real and artificial timing errors could be identified with a high accuracy.

Properties of the Algorithm

In this section, the following properties of the algorithm HyBUTLA are given: *(i)* the algorithm runs in polynomial time, *(ii)* the algorithm learns the correct automaton given a sufficient number of samples, i.e. they identify in the limit (compare also (Gold 1978)), and *(iii)* the presented bottom-up strategy has a better runtime behavior in the average than the previous top-down approaches.

Polynomial Runtime Behavior

Theorem 1. *The algorithm HyBUTLA (algorithm 1) runs in $O(n_0^3)$ where n_0 denotes the number of nodes in the original prefix tree.*

Proof: *Step (1) of the algorithm runs in $O(n_0)$ since the number of events is less than the number of observations. Step (2) also uses every observation once, i.e. it runs in $O(n_0)$. The algorithm HyBUTLA from algorithm 1 compares in step (3) a maximum of n_0^2 nodes. In each merging step for nodes v, w, their both subtrees have to be accessed by the algorithm compatible and determinized.* □

Identification in the Limit

If the number of observations (including negative examples) is constant, Gold proved in (Gold 1978) that the identification of a Deterministic Finite Automaton (DFA) of the given size $k \in \mathbf{N}$ is NP-complete, i.e. no efficient algorithms exist—under the assumption that $P \neq NP$. However, Oncina and Garcia showed in (Oncina and Garcia 1992) that an efficient algorithm exists when the number of observations is not constant but can grow during an algorithm's runtime,—this is called identification in the limit. In this context, an algorithm is efficient when it has a runtime polynomial in the number of observations and the number of observations is polynomial in the size of the optimal automaton.

Definition 2. *(Identification in the limit) Let $\mathcal{O}$ be a set of observations for which an automaton $\mathcal{A}$ should be identified. An automaton learning algorithm identifies $\mathcal{A}$ in the limit if $\mathcal{A}$ is found based on observation $\mathcal{O}, |\mathcal{O}| > n$ for some $n \in \mathbf{N}$.*

It has been proven that stochastic DFAs (Carrasco and Oncina 1999; Thollard, Dupont, and de la Higuera 2000) and also stochastic deterministic timed automata with one clock (Verwer 2010) are identifiable in the limit in polynomial time. Such stochastic automata use only positive observations.

For hybrid automata, the papers (Henzinger et al. 1998; Alur and Dill 1994) give decidability preconditions: *(i)* decoupled output variables, *(ii)* initialization after flow changes, *(iii)* one clock. These conditions are fulfilled here: *(i)* output variables are decoupled, *(ii)* Θ functions implicitly reinitialize variables, *(iii)* only one clock is used.

For the HyBUTLA algorithm, it can be said that as long as the Θ functions are approximated sufficiently well by some regression method in polynomial time, and as long as a single clock is used, HyBUTLA identifies the hybrid system behavior model in the limit in polynomial time. The polynomial runtime behavior has been proven above, so only the identification in the limit has to be shown:

Theorem 2. *Let n_0 be the number of nodes in the prefix tree and a the number of nodes in the correct hybrid timed automaton $A = (S, s_0, F, \Sigma, T, \Delta, Num, c, \Theta)$, i.e. $a = |S|$. Then A is identifiable in the limit.*

Proof: *The analysis follows (Carrasco and Oncina 1994). Generally speaking, two errors might occur when learning with the HyBUTLA algorithm:*

Type I: Two equivalent nodes are not merged.
Type II: Two non-equivalent nodes are merged.

Type I errors can only occur when for two equivalent nodes or their transitions the term $\left|\frac{f_0}{g_0} - \frac{f_1}{g_1}\right|$ has been larger than the error margin

$$\epsilon_\alpha = \sqrt{\frac{1}{2}\log\left(\frac{2}{\alpha}\right)}\left(\frac{1}{\sqrt{g_0}} + \frac{1}{\sqrt{g_1}}\right) \tag{2}$$

(see algorithm 2 and equation (1)). The probability of such an error is α since the null hypothesis for the Hoeffding bounding test is the incompatibility. $(n_0 - a)(|\Sigma| + 1)$ such tests are performed—one for each node merging and one for each corresponding transition. By choosing α as a sufficient small number, the probability of type I errors can be kept small.

Concerning the approximation speed, for a given ϵ_α, the error probability α decreases with $O(\frac{1}{2^{\tilde{n}}})$ where $\tilde{n}$ denotes the number of observations per states of the optimal automaton $\frac{n_0}{a}$. To see this, equation 2 has to be solved for α and g_0 and g_1 have to be set to $\frac{n_0}{a}$.

Presuming that type I errors can be neglected, type II errors result in an automaton that is a subset of the correct automaton. So one of $\frac{a(a-1)}{2}$ compatibility checks have failed where each such check comprises $(|\Sigma| + 1)$ tests of the Hoeffding bounding. Presuming again that g_0 and g_1 from the Hoeffding test (equation (1)) grow linearly with the number of observations, the term

$$\frac{a(a-1)}{2} \cdot (|\Sigma| + 1) \cdot \sqrt{\frac{1}{2}\log\left(\frac{2}{\alpha}\right)}\left(\frac{1}{\sqrt{g_o}} + \frac{1}{\sqrt{g_1}}\right)$$

converges to zero for growing sample sizes. I.e. both errors disappear and the algorithm identifies therefore the correct automaton in the limit. In analogy to the approximation speed analysis from above, again the convergence speed to zero of the error probability is $O(\frac{1}{2^{n_0/a^5}})$. □

Analysis of the Bottom-Up Strategy

Previous algorithms for learning (timed) automata such as (Thollard, Dupont, and de la Higuera 2000; Carrasco and Oncina 1999; Verwer 2010) work in a top-down order. Again, the order of the node is defined by the lexicographic order of the shortest sequence of events leading to a node in the PTA. The algorithm HyBUTLA works in a bottom-up order (see also figure 4). Because each compatibility check for nodes v, w comprises a recursive checking of both subtrees of v and w, HyBUTLA has the advantage of handling smaller subtrees near to the root of the original prefix tree.

In the worst case—no merging of nodes—this different order makes no difference for the runtime behavior. But in most cases, a significant runtime improvement can be seen whereat the degree of improvement depends on the structure of the prefix tree and on the structure of the final automaton.

Here we give a theoretical runtime comparison for an important class of problems: the plant learning problem. When observing plants, we usually have sequences of, more or less, similar lengths—corresponding to a plant cycle. I.e. the prefix tree resembles a complete tree. And the final automaton usually resembles a sequence of states—corresponding to the states of the plant cycle.

Definition 3. *(Plant Learning Problem) In the plant learning problem, the prefix tree acceptor is a complete tree of a fixed node out-degree of $k \in \mathbf{N}$. The optimal merged automaton is a single sequence of states.*

Theorem 3. *For plant learning problems, previous (top-down) algorithms run in $O(n_0^2 \log n_0)$ whereas HyBUTLA runs in $O(n_0)$ where n_0 denotes the number of nodes in the original prefix tree.*

Proof: *The $O(n_0^2 \log n_0)$ results from the sum of subtree sizes for all compatibility checks: $\sum_{i=0}^{n_0-1}\sum_{j=0}^{i-1}(\log(n_0 - i) + \log(n_0 - j))$ where the numbers correspond to the lexicographic node order. HyBUTLA merges two nodes in each compatibility check, so the subtrees have always the height 1. This results in a linear run time behavior.* □

Empirical measurements (see figure 6) show a rather $O(n_0^2)$ runtime behavior. Here, the Reber grammar was used again to compare the runtime. Both algorithms run in n_0^2 in average. However, the bottom-up algorithm works faster than the top-down approach by a factor of ≈ 3.

Summary and Future Work

The automatic learning of models helps to solve the key problem of model-based development of technical systems: The modeling bottleneck—caused by high modeling efforts and the lack of domain experts. In this paper, HyBUTLA, the first algorithm for the learning of hybrid timed automata is presented. A new taxonomy of models and learning algorithms is used to outline the importance of this class of

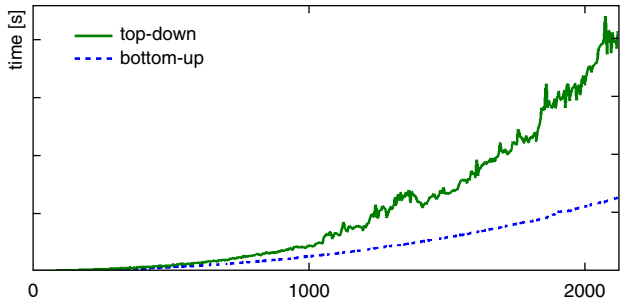

Figure 6: Runtime behavior of the algorithms based on bottom up and top down strategy.

models for the analysis of production plants. HyBUTLA is then analyzed empirically and theoretically. As future work, a better identification algorithm for hybrid states will be developed, fixed time intervals will be replaced by PDFs, and the algorithms will be applied to further plants.

References

Alur, R., and Dill, D. 1994. A theory of timed automata. *Theoretical Computer Science* vol. 126:183–235.

Buede, D. 2009. *The Engineering Design of Systems: Models and Methods*. John Wiley & Sons.

Carrasco, R. C., and Oncina, J. 1994. Learning stochastic regular grammars by means of a state merging method. In *GRAMMATICAL INFERENCE AND APPLICATIONS*, 139–152. Springer-Verlag.

Carrasco, R. C., and Oncina, J. 1999. Learning deterministic regular grammars from stochastic samples in polynomial time. In *RAIRO (Theoretical Informatics and Applications)*.

EnHiPro. Energie- und hilfsstoffoptimierte Produktion. www.enhipro.de. Laufzeit 06/2009-05/2012.

EWOTeK. Effizienzsteigerung von Werkzeugmaschinen durch Optimierung der Technologien zum Komponentenbetrieb. www.ewotek.de. Projektdauer: 01.07.2009 - 30.06.2012.

Fawcett, T. 2006. An introduction to roc analysis. *Pattern Recogn. Lett.* 27:861–874.

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

Group, H.-L. 2006. MANUFUTURE - Strategic Research Agenda. Technical report, European Commission.

Henzinger, T. A.; Kopkeb, P. W.; Puria, A.; and Varaiya, P. 1998. What's decidable about hybrid automata? *Journal of Computer and System Sciences* 57(1).

Isermann, R. 2004. Model-based fault detection and diagnosis - status and applications. In *16th IFAC Symposium on Automatic Control in Aerospace*.

Maier, A.; Vodenčarević, A.; Niggemann, O.; Just, R.; and Jäger, M. 2011. Anomaly detection in production plants using timed automata. In *8th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*.

Markou, M., and Singh, S. 2003a. Novelty detection: A review - part 1. Department of Computer Science, PANN Research, University of Exeter, United Kingdom.

Markou, M., and Singh, S. 2003b. Novelty detection: A review - part 2. Department of Computer Science, PANN Research, University of Exeter, United Kingdom.

Oncina, J., and Garcia, P. 1992. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, volume 1 of *Machine Perception and Artificial Intelligence*. World Scientific,.

Pethig, F., and Niggemann, O. 2012. A process data acquisition architecture for distributed industrial networks. In *Embedded World Conference*.

Preuße, S., and Hanisch, H.-M. 2009. Specification of technical plant behavior with a safety-oriented technical language. In *7th IEEE International Conference on Industrial Informatics (INDIN), proceedings pp.632-637, Cardiff, United Kingdom*.

Reber, A. S. 1967. Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior* 6(6):855 – 863.

Schmidt, M., and Lipson, H. 2009. Distilling free-form natural laws from experimental data. *SCIENCE* 324.

Stein, B. 2001. *Model Construction in Analysis and Synthesis Tasks*. Habilitation, Department of Computer Science, University of Paderborn, Germany.

Thollard, F.; Dupont, P.; and de la Higuera, C. 2000. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proc. 17th International Conf. on Machine Learning*, 975–982. Morgan Kaufmann.

VDI/VDE. 2009. Automation 2020 - bedeutung und entwicklung der automation bis zum jahr 2020.

Verwer, S.; de Weerdt, M.; and Witteveen, C. 2010. A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In Sempere, J., and Garcia, P., eds., *Grammatical Inference: Theoretical Results and Applications*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

Verwer, S. 2010. *Efficient Identification of Timed Automata: Theory and Practice*. Ph.D. Dissertation, Delft University of Technology.

Vodenčarević, A.; Büning, H. K.; Niggemann, O.; and Maier, A. 2011. Using behavior models for anomaly detection in hybrid systems. In *Proceedings of the 23rd International Symposium on Information, Communication and Automation Technologies-ICAT 2011*.

Witten, I., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

Wymore, A. 1976. *Systems Engineering for Interdisciplinary Teams*. New York: John Wiley & Sons.

Zeigler, B.; Praehofer, H.; and Kim, T. 2000. *Theory of Modeling and Simulation*. New York: Academic Press.