Intelligent control systems
Learning, interpreting, verification

Lin, Qin

DOI
10.4233/uuid:7b17a968-1414-4b84-bbf3-9a0c1197e1fd

Publication date
2019

Document Version
Final published version

Citation (APA)
Lin, Q. (2019). *Intelligent control systems: Learning, interpreting, verification*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:7b17a968-1414-4b84-bbf3-9a0c1197e1fd

Intelligent Control Systems

Learning, Interpreting, Verification

Intelligent Control Systems

Learning, Interpreting, Verification

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus prof.dr.ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates
to be defended publicly on
Thursday 5 September 2019 at 10:00 o'clock

by

Qin Lin

Master of Engineering in Control Theory and Control Engineering, Tongji University, China
born in Foochow, China

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus	chairperson
Prof.dr.ir. J. van den Berg	Delft University of Technology, promotor
Dr.ir. S.E. Verwer	Delft University of Technology, copromotor

Independent members:

Prof.dr. C. Witteveen	Delft University of Technology
Prof.dr. F. W. Vaandrager	Radboud University Nijmegen
Prof.dr. J. M. Dolan	Carnegie Mellon University, USA
Prof.dr. A. P. Mathur	Singapore University of Technology and Design, Singapore Purdue University, USA
Dr. H. H. Hansen	Delft University of Technology

This thesis was partially supported by NWO.

Nederlandse Organisatie voor Wetenschappelijk Onderzoek

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

SIKS Dissertation Series No. 2019-23 The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover design by Peilin Lin

For my mother

Acknowledgements

Chinese academics often find themselves pursuing what is sometimes called the "three immortalities": moral *worth*, significant *work*, and persuasive *word* (三不朽：立德、立功、立言).[1] These values act as a loadstar, guiding scholars as they strive to achieve meaningful lives. Throughout this journey, I have relied on my faith to keep me grounded. Buddhism, Daoism, and Confucianism have guided me through stress. I am the person I am today thanks to their nourishing influence. The hope driving the pursuit the *work* and the *word* is that the fruits of our labor prove to be lasting contributions to our field and our communities, outliving us and becoming a foundation upon which the next generation of researchers can stand. My greatest ambition is that this thesis provides another scholar with a useful new perspective and foundation that they can build upon in their own work. The far-reaching goal of "immortality" seems achievable in such a small way. For me, research has been a labor of love bringing me simple joys and self-satisfaction.

TO MR. S. T. COLERIDGE

Midway the hill of science, after steep
And rugged paths that tire the' unpractised feet,
A grove extends; in tangled mazes wrought,
And filled with strange enchantment:—dubious shapes
Flit through dim glades, and lure the eager foot
Of youthful ardour to eternal chase.

Anna Laetitia Barbauld (1743–1825)

I would like to express my deep gratitude to my daily supervisor, Dr. Sicco Verwer. With his patient guidance, I first made a start in the field of machine learning. It was his encouragement that first brought me to the research field combining machine learning and verification in which I hope to develop a new line of research. There are much more than these two things I need to thank him in the past four years staying together with him as colleagues and friends.

I want to express my sincere appreciation to my promotor Prof. Jan van den Berg. Jan always has lots of experiences and stories to inspire people around him. He pushed me to think deeper in research and taught me how to be a better person. He is the first person to ever call me a scientist, and I will never forget the pride those words made me feel. He encouraged us to cherish our life by telling his living and teaching experience in Africa.

[1] called 3W in the essay, *Immortality–My Religion*, written by Hu Shih

I wish to acknowledge the colleagues I work with on the autonomous driving topic - Prof. Jun Wang and Yihuan Zhang at Tongji University, China. Thanks for their trust and interest in my research and for offering me the opportunity to apply my techniques. I am particularly grateful for the opportunity given by Prof. John M. Dolan at Carnegie Mellon University. Thanks for his interest in my work on verifiable learning-based models. I am honored to have the opportunity to continue my work with you. I look forward to exploring more exciting research in this field with you.

I would like to thank the collaborators from SUTD I worked with on the CPS security topic: Prof. Aditya Mathur, Adepu Sridhar, and Prof. Robert Kooij. I always remember Aditya's encouraging words at the conference. I am very impressed and inspired by his ambitions and diligence. It is sad that I have not found an opportunity to work with him more closely. But I hope to continue the collaboration in the near future. Thanks to Adepu for teaching me lots of knowledge about CPS security and the SWaT testbed. Thanks to Robert for connecting me with SUTD and many pieces of advice he offered.

I thank all other faculty colleagues: Pieter, Stjepan, Christian, Zeki, Inald, Jan, Jos, and Phil in Cyber Security group for introducing me to a world knowledge well beyond the scope of this thesis. I appreciate every joyful day spent with my groupmates: Nino, Chris, Azqa, Mark, Vincent, Harm, Oguzhan, Majid, Gamze, Chibuike, Laurens, and Zhijie. Many thanks to our secretary, Sandra Wolf, for all help related to my work and my personal life. Special thanks go to Chris, Sara, and Prof. John Dolan for many rounds of proofreading. Sara offered great help to rephrase many Eastern philosophical sayings to become understandable for Western people. Many thanks to Laurens, Sicco, Mrs. Ankie Verwer, and Mr. Piet Verwer for their kind help with the Dutch translation.

I don't know who I would be without my friends: shibei Wu, Miao Sha, Xucheng Yin, Cong Liu, Shiwei Bao, Kaixin Ding, Yi Guo, Jun Liu, Xiaoran Liu, Jing Wang, Zijin Ren, Yazhou Yang, Ding Ding, Lingling Lao, Yu Xin, Yingqian, Xiuxiu Zhan, Zixuan Zheng, and Zina Wang. Special thanks go to my piano teacher Jia Qu for the joy of playing music with her and for the interesting discussion about music and math harmony. Special thanks go to Peilin for designing the lovely cover of this thesis. I thank Shubin for the kind help of cover photo adjustment.

I would like to thank Prof. Frits Vaandrager, Prof. Cees Witteveen, and Dr. Helle Hansen for their invaluable comments and kind service as committee members.

I still remember the rainy day my parents and my grandmother said goodbye to me when I was leaving my hometown for the Netherlands. I do hope time slows down a bit more then I will share more joyful moments with you in the future.

My last and deepest gratitude goes to my wife, Hui. She always stands by me and remains tolerant through all my absences and impatience. She was the most worried person when I went travelling far from home, when I got sick, and when I worked too hard. She gave me support and help, discussed ideas and prevented me from taking several wrong turns as my best friend and mentor. Her adventurous spirit and her flexibility for starting new lives in multiple countries encouraged and inspired me. She is my greatest love and the partner for life anyone could ask for.

Karma in Buddhism is a spiritual principle stating that good intent and good deeds contribute to good future lives. Buddhist cosmology says there are countless Buddhas and countless Sahasra (meaning "one thousand"; in modern parlance it is roughly a "so-

lar system") worlds. I must have good intent and good deeds in my past life to meet all of you. I cherish the fortune to meet you in the same Sahasra among the countless universes.

Qin Lin
呼牛斋,[2] the U.S.
Aug. 2019

[2]The name of my reading room, adapted from the word 呼牛唤马(Hu niu huan ma) in the book 《庄子•天道》(Zhuangzi, The Way of Heaven). It is a Chinese idiom meaning that it doesn't matter you call me a cow or a horse. It's a metaphor representing the philosophy that we should never take others' insults or praises seriously.

Contents

List of Figures

List of Tables

1

Introduction

1

1.1. MOTIVATION FOR HYBRID SYSTEM LEARNING

Autonomous vehicles (AVs) are on the way to take over our daily driving tasks. People are endowing the machine with human-level driving intelligence to perceive the surrounding traffic environment, make reasonable decisions, and control the vehicle. Human driving behaviors are, however, highly complex, making them difficult to understand. Obtaining accurate first-principle dynamical models needed to describe them is often difficult. Alternatively, we can use an *intelligent controller* capable of learning and mimicking a human driver that generates this behavior. A human driver can serve as a teacher to "teach" such a controller how to drive, by providing a large amount of driving data as input and control actions as output.

The essential task in such a system is to establish a "mapping" (actually a stimulus-response relation) from observations of the traffic environment measured from sensors, to control actions executed by human drivers. In order to achieve this goal, an obvious and trivial solution would be to pre-program rules by enumerating all possible traffic situations and applying the corresponding reactions. However, it is impractical to realize a complete rule-based system from a highly complex traffic environment. An intelligent controller automatically learns the underlying driving rules and continuously improves its performance, e.g., by minimizing the difference between its own and human driving behaviors.

A key characteristic of human driving behaviors or more general human control behaviors is their hierarchical or hybrid property (Buntins et al., 2013). Imagine that a driver is attempting to merge into an adjacent lane. The complete maneuver consists of three stages: first, the driver is *following* the leading vehicle in his own lane; second, he is *shifting* the vehicle to the target lane; finally, he continues to follow the leading vehicle in the new lane. It is evident to observe the high-level switching behaviors such as car-following and lane change. In addition, in each stage, the continuous dynamics in terms of longitudinal and lateral movement are observed in the low-level control. *The first goal of this thesis is learning-related: designing a proper intelligent controller to capture such heterogeneous and hybrid behaviors*, which will be discussed in Chapters 3 and 4.

Safety is an important concern for promoting a wide adoption of autonomous vehicles. The intelligent controller normally serves as a "black-box" impeding us from having insightful ideas about whether and how it reacts in different situations. A strong demand for the intelligent controller is the full exposure of its model, which should be understandable and verifiable for human beings. *The second goal of this thesis is safety-related: the intelligent controller should be both explainable and safe*, which will be discussed in Chapters 5, 6, and 7.

1.1.1. COMPLEXITY BOTTLENECK OF CONVENTIONAL CONTROLLER DESIGN

A controller is a device that adjusts output control signals sent to an actuator based on the sensor signal to change the condition of a plant. Figure 1.1 shows a diagram of a typical closed-loop system in classical control theory. Take a car's cruise controller (CC) for example (Nice, 2001). The controller (C) is a device designed to maintain vehicle speed at a constant desired or reference speed (**r**) provided by the driver. The plant (P) is the car, and the whole system consists of the car and the cruise controller. The system output ($\mathbf{y}$) is the car's speed, and the control command denoted by the variable ($\mathbf{u}$) is the engine's throttle position. The block *Measurement* usually serves as a transducer, i.e., it transforms the kinetic signal (car's speed) into a digital signal for a further calculation. The key concept of feedback control is that the input of the controller is actually the difference ($\mathbf{e}$) between the system's output (the current speed) and the reference (the desired speed), i.e., $\mathbf{e} = \mathbf{y} - \mathbf{r}$. An intuitive control law of the controller is: if the output speed is larger than we desire, the controller tries to decrease it accordingly. In practice, we need a mathematical formula as an analytical tool to precisely describe such a control law.

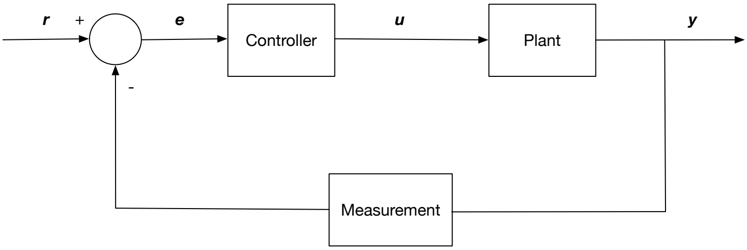


Figure 1.1: Closed-loop system in conventional control theory (Franklin et al., 1994)

The main idea of the conventional controller design is building rigorous mathematical models to describe the dynamics of the controller, the plant, and the measurement, respectively. A differential equation, a transfer function, or a state space equation are the three most commonly used mathematical models (Polderman and Willems, 1998). Accurate physical descriptions are vital to design such models. For example, Newton's laws and Kirchhoff's laws are applied to obtain differential models in mechanical systems and electrical systems, respectively. An example *state space model* of a system can be defined in the following set of equations:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t),\ \mathbf{u}(t),\ t), \quad \mathbf{x}(t_0) = \mathbf{x_0} \tag{1.1}$$

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t),\ \mathbf{u}(t),\ t) \tag{1.2}$$

where $\mathbf{x}$ is a set of state variables of the system, $\mathbf{u}$ the input control variable, $\mathbf{x_0}$ the initial state, $\mathbf{y}$ the output variable. Note that, in the simple cruise control example, $\mathbf{y}$ and $\mathbf{x}$ are both equal to the car's speed. Many differential equations of interest in continuous-time models do not have a closed-form solution. Computers can aid to solve these equations numerically. Therefore, an alternative form known as *difference equations* replaces differential equations in discrete sampling time as follows:

$$\dot{\mathbf{x}}(t+1) = \mathbf{f}(\mathbf{x}(t),\ \mathbf{u}(t),\ t),\ \mathbf{x}(t_0) = \mathbf{x_0} \tag{1.3}$$

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t),\ \mathbf{u}(t),\ t) \tag{1.4}$$

The control laws (algorithms and/or mathematical models) are realized via software or hardware design. The first work of mathematical modeling-based controller design

1

is dated to 1867 (Clerk, 1867; Antsaklis et al., 1993). In this work, the differential equations were used to model the dynamics and to analyze the stability of a flyball governor, controlling the speed of an engine by regulating the amount of fuel admitted, so as to maintain a near-constant speed, irrespective of the load or fuel-supply conditions.

To make a solid and intuitive example for a vehicle system, let us consider a dynamical control law in an adaptive car-following scenario.

$$\dot{v_f} = C_1 \cdot (v_l - v_f) + C_2 \cdot \big((x_l - x_f) - D\big) \tag{1.5}$$

and

$$D(t) = \alpha + \beta \cdot v_f \tag{1.6}$$

where x_f and v_f are state variables of the host vehicle, and x_l and v_l are observations from the environment (namely the lead vehicle in this case). These can be considered as uncontrolled input, and D is the desired relative distance. The control output is quite straightforward as a linear combination of relative speed, relative distance, and the host vehicle's speed. Intuitively speaking, acceleration as a large control action is needed, when the relative speed and the difference between relative distance and desired relative distance are positive and large. Conversely, the controller conducts deceleration when the aforementioned two difference values are negative. The desired distance is linearly dependent on the current speed of the ego vehicle (i.e., our car). For example, we need a relatively large desired relative distance when we are driving fast to enhance the safety.

Note that, in this case, the form of the equations that map the observations x_l, v_l, x_f, v_f to the control output $\dot{v_f}$ is assumed to be known *a priori*; only the parameters of the equations are unknown.

Mathematically modelling, as a first-principle design, has been a bottleneck of the conventional controller design due to increasing complexities of control systems. A more flexible approach is needed to model the control behaviors by approximating the input and output mapping without understanding the detailed physical processes.

1.1.2. INTELLIGENT CONTROL SYSTEM: OPPORTUNITIES AND CHALLENGES

The notion of intelligent control systems (ICS) was developed in the work of K.S. Fu in the 70's (Fu, 1970; Antsaklis, 2001), where actually the author used another term, "learning control systems". We use the definition of intelligent control systems in (Antsaklis, 2001):

> *Intelligent controllers* can be seen as machines that emulate target faculties via learning from large amounts of data, and safely conduct tasks in a highly uncertain environment.

At a minimum, intelligence requires the ability to sense the environment, to make decisions and to take control actions. Note that in conventional control, the input of reference and feedback can be seen as simplified environmental inputs in the ICS. The higher levels of intelligence may include the ability to recognize objects and events, to represent knowledge in a world model, and to reason and plan for the future. Conventional control usually serves as a low-level task in the intelligent control system.

The definition of ICS is variegated in the literature. A general consensus is that learning plays a fundamental role in each level of the intelligent controller. Learning was viewed as the estimation or successive approximation of the unknown quantities of a function. There are many areas in a control system where learning can be used (Antsaklis, 2001): 1. Learning about the plant and even dealing with the plant's changes and then deriving new plant models. 2. Learning about the environment; this can be done using methods ranging from passive observation to active experimentation. 3. Learning about the controller; in the context of supervised learning, this is about how to behave in a dynamical environment from the "demonstration" of the teacher. This dissertation mainly deals with learning of the environment and the controller.

Depending on whether a teacher exists to guide the learning, the learning can be classified as *supervised* and *unsupervised*. Supervised learning supposes that a teacher is available to give an answer about the desired output of the system or optimal control action. For unsupervised learning, also called learning from experience, the learning is directed by some performance measure through trial and error.

The teacher in a supervised learning setting does not have to be a human. Both *animated systems* such as human beings and *unanimated systems* such as industrial control systems can serve as supervisors in different application scenarios.

Developing an ICS is an interdisciplinary research work involving knowledge from artificial intelligence, control theory, and computer science. It is challenging due to many open problems when developing a system with a high degree of autonomy and intelligence. Indeed, it is not possible to address all of these questions using techniques introduced in this dissertation. Motivated by the following *key concerns* about the fundamental requirement of designing an ICS, we propose techniques that offer solutions of practical avail.

- **Learning-related** (about the first goal mentioned in Section 1.1)

  1. *Intelligent control systems should have a proper learning ability:* (discussed in Chapters 3, 4) Machine learning is becoming a powerful technique in artificial intelligence to devise complex models and algorithms that lend themselves to prediction. In this dissertation, we focus on supervised learning. The computer or agent is fed example inputs and their desired outputs, given by a "teacher", and the goal is to learn a generalized rule that maps inputs to outputs. The standard supervised learning approach usually makes an independent and identical distribution (i.i.d.) assumption, e.g., the mapping pairs of states and control actions are independent. However, in many application cases, the demonstration of the teacher is essentially a sequential decision making process, where the i.i.d. assumption does not hold any more. Therefore, the first question is how to learn a proper sequential model from a demonstration.

  2. *Intelligent control systems should have hierarchical functionality:* (discussed in Chapters 3, 4) In this dissertation, the hierarchical functionality refers to *hybrid* behavior involving discrete and continuous dynamics. The motivation is twofold:

     (a) Transparent and precise modelling: Recall the example of the merge lane driving scenario. The driver shows *heterogeneous* behaviors in different states of lane keeping and lane changing. For the existing intelligent control systems such as neural networks, such a composition of discrete and continuous dynamics is unfortunately vague. Instead, modelling in a piece-wise manner based on similarities of conditions helps us obtain a more precise and more insightful description for heterogeneous behaviors.

(b) Hierarchical tasks: The three levels of a hierarchical ICS architecture based on a "divide-and-conquer" spirit are the Execution Level (EL), the Coordination Level (CL), and the Management Level (ML) (Antsaklis, 2001). EL involves conventional control algorithms, while the highest ML involves only higher- level, intelligent, decision-making methods. The CL is the level providing the interface between the actions of the other two levels. It uses a combination of conventional and intelligent decision-making methods. A simplified lane change example is presented here to clarify the responsibilities of each level in an autonomous driving car. The car abstracts and understands the traffic environment using classification. The reasoning and planning can be done in a high level and make an optimal decision such as lane change. The task is then sent to the middle level to make an optimal plan for the lane change. The lowest level conducts the real-time control of the vehicle to continuously adapt the (lateral and longitudinal) position to the target lane on the basis of conventional vehicle dynamic control.

- **Safety-related** (about the second goal mentioned in Section 1.1)

3. *Intelligent control systems should behave socially:* (discussed in Chapter 5) An ICS usually interacts with other agents involved. An example is the interaction of autonomous vehicles with other human-controlled vehicles. The maneuver of lane changes from human drivers is sometimes conducted without signaling. Predicting the intention of a lane change reduces the risk of collisions in these cases. The control action of the ego vehicle is performed in a more "conservative way" to handle the possible cut-in behavior.

4. *Intelligent control systems should be self-diagnosable:* (discussed in Chapter 6) Fault diagnosis and alarm functionality need to be accomplished in an ICS because the system needs to conduct adaptive control reconfiguration and maintenance scheduling in a highly uncertain environment. A new perspective on this problem comes from the growing threats of cyber attacks to safety-critical industrial control systems. A concrete example concerns the physical cyber attacks in supervisory control and data acquisition (SCADA) systems, which are commonly used in industrial control systems.

   The physical cyber attacks often refer to an attacker who tries to falsify the reading of sensors or actuators and to disrupt the state of the system. Such attacks would cause catastrophic consequences in critical infrastructure such as power plants (Falliere et al., 2011; Case, 2016) and water treatment systems (Slay and Miller, 2007). A "good" model that approximates the original control system is essential to profiling all legitimate behaviors and detecting significant deviations from this model caused by an intrusion.

5. *Intelligent control system should be verifiable:* (discussed in Chapter 7) A general ICS only captures a mapping from environment to control actions in a simplified “black-box" fashion without any insightful understanding of the system itself (Mühlegg et al., 2015). The computation and learning procedure should be traceable in an explainable ICS model. As a result, it helps people to discover how an intelligent controller makes its decisions and to do troubleshooting when faults occur. Moreover, learning-based controllers have much fewer theoretical performance guarantees than rigid mathematical modeling of conventional control. Such guarantees are crucially needed in safety-critical infrastructures such as water, power grid, and nuclear systems.

1.1.3. RELATED WORK

Learning for intelligent control has attracted many researchers in the past decades. However, few works focus on learning hybrid control systems. Reinforcement learning (RL) uses a trial-and-error principle of learning in environments without supervisors. The control policy in RL maximizes the numerical reward from the environment. The main drawback of RL is its inefficiency of learning (Schaal, 1999).

Another drawback of RL is that the reward function is not trivial to design in practice. A potential solution is inverse reinforcement learning (IRL). The idea is that the demonstrator is assumed to perform optimal control actions. The first step of learning is obtaining an approximation of the reward function from the demonstrator using base functions such as polynomial, Fourier, etc. Then the learner seeks to maximize the reward like the task in RL. The representative works include apprenticeship learning (Abbeel and Ng, 2004), maximum margin planning (Ratliff et al., 2006), and structured classification (Klein et al., 2012). IRL is a kind of indirect supervised learning sitting between standard direct supervised learning and unsupervised learning.

There are two classes of approaches on inferring hybrid automata. The first class is *language learning* (Niggemann et al., 2012; Medhat et al., 2015). First, the continuous signal is segmented using signal processing; then the symbolic strings are used for inferring a finite state machine; last, differential equations in the modes, namely the states in the FSM are identified from the continuous signal. The second class is *numerical model learning*. State space equations are common tools for learning a Markov jump system. In order to optimize using expectation maximum (EM) or maximum likelihood estimation (MLE), some assumptions about the underlying formula are made. For example, (Summerville et al., 2017) assumes linear dynamics in the modes, and (Ly and Lipson, 2012; Santana et al., 2015) assume the number of modes is known in advance.

Owing to its logical and graphical features, a finite state automaton is highly insightful for human beings to read and understand the internal mechanism of the studied systems’ behaviors, which has gained great success in many application domains (Hammerschmidt et al., 2016; Pellegrino et al., 2017b; Liu et al., 2017b). One reason is its versatility, e.g., it can be deterministic, nondeterministic, probabilistic and hybrid. The states can be observable or hidden. It is able to play key roles in multiple sequential tasks such as an acceptor in a sequential classification problem, a transducer in sequence-to -sequence problems, and a generator as a generative model of sequences (Castro and Gavalda, 2016). Another advantage is that many algorithmic problems are computationally feasible for an automaton. The determinization, minimization and equivalence solidify the foundation of automata learning. The set-theoretic and linear-algebraic operations make the verification of hybrid automata possible. Learning an automaton from a supervisor for a control task has been suggested in the literature (Martins et al., 2001, 2002). However, it is rare to see a systematic work discussing learning and verification of an intelligent controller using a hybrid automaton.

1.2. CONCEPTUAL APPROACHES

A diagram of an intelligent controller is shown in Figure 1.2: The supervisor provides demonstrations of actions output in its environment. The intelligent controller is capable of mimicking the supervisor's behavior by learning a sequential model. Besides that it can also learn the model for the environment and other agent. The intention prediction of other agents is realized in the perception part. The self-diagnose part checks whether the state of the system is disrupted by attacks. The self-verification component automatically checks the safety specification in each state.

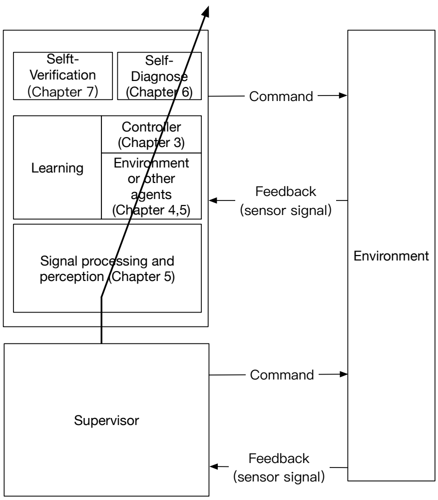


Figure 1.2: The system hierarchy of the intelligent controller studied in this dissertation. The corresponding research content of each chapter is also annotated in each component. The arrow depicts that the intelligent controller is able to function like its supervisor.

To achieve the functionalities of an intelligent controller mentioned above, the technology roadmap is briefly summarized as:

1. **Chapter 3:** The teacher's sequential demonstration is considered as a linguistic source of control actions. We focus on learning an automaton to represent an underlying language model in this dissertation. To deal with the hybrid characteristics in the control actions, we first investigate a *composed* type of learning hybrid automata. The discrete events are first abstracted from similar environmental inputs. Second, they are used for learning the structure of a hybrid automaton. The

numeric data are used for identification of the parameters in the differential equations defining the numerical input and output mapping in each mode. State clustering is introduced to abstract the automaton model and reduce the number of modes. This makes a trade-off between prediction accuracy and model complexity.

2. **Chapter 4** Another novel *inline* type of learning hybrid automaton is proposed that simultaneously considers discrete (abstraction from raw numerical data) and continuous data (first-order differential information in the raw numerical data). During the state machine learning procedure, the similarity of the first-order difference (described in the state) and the symbolic event are checked. The model is used for learning an auto-regressive model.

3. **Chapter 5** To deal with the interaction with other participating agents, a non-deterministic automaton is learned as a probabilistic classifier for behavior recognition. The classification results are integrated as the stochastic input to the optimization task of model predictive control (MPC) in the ego agent.

4. **Chapter 6** In the self-diagnose task level (cf. Figure 1.2), another way to deal with the mapping of multiple inputs and outputs in a high-level behavior learning process is proposed. A novel combination of automata learning and Bayesian network learning is investigated to deal with this problem, where an automaton is used to represent the dynamics in the output of a system, and the dependency among sensors and actuators is learned by Bayesian network inference.

5. **Chapter 7** Reachability analysis is leveraged to verify the safety specification of a system. The bad state is identified where collision happens. An imitation learning-based controller is learned from data generated by a human. A hybrid model checking tool is used for the safety verification of the data-driven controller.

1.3. CONTRIBUTIONS

This thesis makes four major contributions to the field of machine learning and its applications in autonomous driving and the security of industrial control systems. The details of each contribution are also summarized correspondingly.

1. It proposes two novel approaches of learning hybrid automata: *composed* and *inline* algorithms.

   The existing composed approach learns a distinct continuous model in each mode of a hybrid automaton, which introduces high complexities. Our MOHA model is a novel composed approach achieving a trade-off between accuracy and complexity by clustering similar modes (Zhang et al., 2017a,b; Lin et al., 2018b). The model achieves great success in learning car-following behaviors from human driving data, which is potentially used as a data-driven cruise control system.

A novel model called a regression automaton is proposed for extending the semantics of conventional deterministic finite automata (DFA) (Lin et al., 2016). This makes DFAs applicable to general numerical tasks such as time series modeling and prediction. The inline approach is a novel algorithm developed based upon a new heuristic state-merging technique. The new model and the new algorithm together partially inspire the development of an advanced passive automaton learning tool called *flexfringe* (Verwer and Hammerschmidt, 2017). This work makes a contribution to advanced automaton learning algorithms.

2. It develops a safe cut-in-awareness car-following controller in autonomous driving systems.

   We apply a probabilistic automaton learning approach for profiling cut-in (lane change) behaviors of human drivers (Zhang et al., 2018). The lane change intention is computable and predictable from this model. A model predictive control then uses such a stochastic input to achieve a collision-avoidance cruise control. This research will stimulate the further development of advanced driver-assistance systems (ADAS).

3. It proposes the first explainable intrusion detection and localization system.

   We apply timed automata learning for discovering behaviors of sensors in an industrial control system. Bayesian network learning is leveraged to discover the causalities between sensors and actuators. They are combined ino a model called TABOR for detecting anomalies caused by data manipulation of cyber attacks (Lin et al., 2018a). TABOR successfully achieves high detection accuracy and explainability for localizing the faulty components. This research will stimulate the development of methords for protecting safety-critical infrastructure.

4. It presents the first safety-verifiable adaptive cruise control model using a hybrid automaton learned from human driving data.

   We develop a translator called MO2SX filling the gap between MOHA and the state-of-the-art hybrid model checker SpaceEx. A complete framework is therefore available for automatically learning and verifying the safety properties of a cruise controller from human driving data. This framework is generic and extendable to more complex driving behaviors.

1.4. OUTLINE

This thesis is divided into the following chapters:

Chapter 2. This thesis begins with an explanation of variate automata models such as deterministic automata, probabilistic automata, timed automata, and hybrid automata.

Then an extensive survey of related work on hybrid automaton learning is presented. In addition, a gentle introduction about safety verification for hybrid automata is provided.
Chapter 3. In this chapter, the model called multi-mode hybrid automaton (MOHA) is proposed as well as its composed learning algorithm ,including time automaton learning, parameters identification in continuous models, and mode identification by state clustering. MOHA is applied to learning car-following behavior from human drivers.
Chapter 4. A novel hybrid model called *regression automaton* and its inline type of learning are described in this chapter. It is applied to learn an auto-regression model for time series modeling and prediction.
Chapter 5. This chapter first shows how to use non-deterministic automata learning to address the lane change intention prediction problem in autonomous driving vehicles. The intention is used as a stochastic input to an ego vehicle's adaptive cruise controller. The efficiency of this framework is demonstrated in the application of a lane-change-awareness cruise controller design.
Chapter 6. In this chapter, the TABOR model is introduced to combine automata learning and causality inference using a Bayesian network. TABOR is applied to detecting anomalies in a water treatment testbed.
Chapter 7. The translator called MO2SX is first introduced. Extensive experiments in both highway and urban traffic are carried out to verify a data-driven cruise controller based on the MOHA model.
Chapter 8. Concluding remarks are made in this chapter to summarize the contributions made by this thesis. The possible societal impact of this thesis is discussed. Future work and suggestions on both theory and application are provided as well.

2

BACKGROUND

2.1. INTRODUCTION

This chapter contains an explanatory survey of automata models (Sections 2.2, 2.3), automata learning (Section 2.4), and verification of hybrid systems (Section 2.5). In addition, an overview of the state of the art in each of these fields is provided. The survey can be read without substantial prior knowledge of these fields.

The remainder of this chapter is split into three sections, one for each topic. The sections on these topics can be read independently and skipped if necessary. In the main text of this thesis, we refer to the relevant background knowledge from this chapter whenever required.

2.2. TIME-DRIVEN AND EVENT-DRIVEN SYSTEMS

In continuous-state systems the state generally changes as time changes, as shown in Equation 1.1 and Equation 1.2. Similarly, in discrete-time models, which are shown in Equation 1.3 and Equation 1.4, with every "clock tick" the state is expected to change. We refer to such systems as *time-driven systems*. In such a system, the state transitions are synchronized by a clock. The clock alone is responsible for any possible state transition. For *event-driven systems*, at various time instants (not necessarily known in advance), some event e "announces" that it occurs. The state evolution depends entirely on the occurrence of asynchronous discrete events.

2.2.1. DISCRETE EVENT SYSTEMS

An *event* is defined as a specific action taken, e.g., pushing the cruise control button on a car. Note that an event may also be the result of several conditions that are suddenly met, e.g., vehicles' relative distance reaches a given value. A discrete event set E contains all events as its elements.

Definition 2.1. *(Discrete Event System)* A discrete event system (DES) is a discrete state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time (Cassandras and Lafortune, 2009).

A DES satisfies the following two properties:

1. The state space is a discrete set.

2. The state transition mechanism is event-driven.

In contrast to DES, a Continuous-Variable Dynamic System (CVDS) refers to the behaviors in Equations 1.1, 1.2, 1.3, and 1.4. A CVDS has the following two properties:

1. The state space is continuous.

2. The state transition is time-driven.

Figure 2.1 shows an example distinguishing the behaviors of a CVDS and a DES from a piece of speed record from a car. The dynamics of the DES can be seen as a piecewise constant function, where the state jumps from one discrete value to another whenever an event takes place. In this case, the event is associated with the state change, e.g., at $t = 46$, the state changes from c to b, where we can say the event $c \to b$ happens. Note that in the definition of a DES, the event can be from any reasonable set predefined for us, e.g., arbitrary input actions, and is not necessarily to be the state change as in this example.

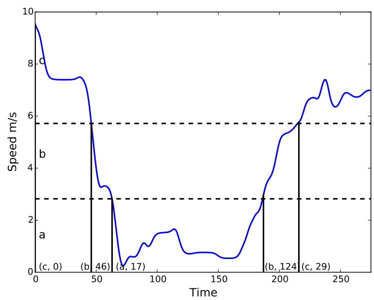


Figure 2.1: A speed record from a vehicle. The value of speed's state space is partitioned into three zones named as a, b, c. In this case the partition is based on Voronoi cells, where in each zone, all its values are closest to its own cluster's centroid. Any clustering or discretization methods can serve as plug-and-play approaches to partition the continuous values here. The time information besides the event is the time difference between successive events.

For a better understanding of the discrete events' behaviors in terms of ordering and timing information of each event, we need a proper representation of the data. A convenient way to describe the timed and logical behaviors of the events in the DES in Figure 2.1 is:

$$(e_1, t_1), (e_2, t_2), (e_3, t_3), (e_4, t_4), (e_5, t_5) = (c, 0), (b, 46), (a, 63), (b, 187), (c, 416)$$

The first event e_1 occurs at t_1, second event e_2 occurs at t_2 and so forth. The sequence is called a *timed string*. The string without time information is called an *untimed string*, and just represents the logical ordering of the events. The set of all possible timed (untimed) strings executed by a DES is called a *timed language* (*untimed language* or *language*). This is because the event set $E = \{e_1, e_2, \cdots, e_n\}$ can be seen as an *alphabet*, and the sequences can be seen as *words*. Additional timed information is sometimes represented as the "lifetime" indicating the elapsed time between successive occurences of each event, as shown in Figure 2.1. The dynamics can be further refined if some statistical information is available. Probability distribution functions can be used in either modelling the lifetime of each event or modelling the state transitions. This results in a *probabilistic timed language*. Language, timed language, and stochastic timed language comprise three levels of abstraction of a DES. The choice of the appropriate level of abstraction depends on the application tasks.

The language-based approach itself is not sufficient to address DES tasks such as simulation, verification, controller synthesis, etc. If a language (e.g., timed language or stochastic timed language) is finite, we could always list all its elements, that is, all the possible strings that the system can execute. Unfortunately, this is unrealistic in the real world. Preferably, we would like to use models that would allow us to represent languages in a manner that highlights the structural information about the system behavior and that is convenient to manipulate when addressing analysis issues. Discrete event modeling formalisms can be untimed, timed, or stochastic, according to the level of abstraction of interest. In this thesis, we will focus on a popular discrete event modeling formalism: the automaton model. In the following subchapters, non-timed automata, timed automata, and stochastic automata are introduced as per the three levels of abstraction of DESs.

2.2.2. NON-TIMED AUTOMATA

As a computation model, an automaton can accept/reject strings, generate strings, or both. Thus, generally we have three types of automata:

1. Generator: the computation machine generates all possible output strings.
2. Acceptor: the computation machine accepts or rejects some input strings.
3. Transducer: the computation machine generates output strings from input strings.

In practice, the generator model can act as a simulation model to generate all valid behaviors of a DES. The acceptor model can be a binary classifier for accepting or rejecting new arriving strings. These two models are normally suitable for autonomous dynamical systems without input. The transducer can deal with the input and output mapping in a DES.

DETERMINISTIC AUTOMATA

We start with the basic model-deterministic finite automaton (DFA). Other much more complex models are built on a DFA. A DFA has a formal definition as follows:

2

Definition 2.2. *(Deterministic finite state automaton, DFA)* A DFA is a quintuple $\mathscr{A} = \langle Q, \delta, \Sigma, q_0, F\rangle$ where Q is a finite set of states, $\delta : \Sigma \times Q \to Q$ are labeled transitions with labels coming from an *alphabet* Σ, $q_0 \in Q$ is the start state, $F \subseteq Q$ is a set of final states.

Note that the transition function of the automaton δ, is also called z partial mapping. The language represented by $\mathscr{A}$ is only the subset of all possible strings Σ^*.

Definition 2.3. A run of a DFA over a string $a_1, a_2, a_3, \cdots, a_n$ is:

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$$

where $\delta(a_i, q_{i-1}) = q_i$ for $i \in \mathbb{N}_+$, $q_i \in Q$, and $a_i \in \Sigma$. The run is valid when $q_n \in F$.

Example 2.1. A simplified cruise controller is illustrated in Figure 2.2 as an example of a DFA. The initial state is the state *off*. The state transition is governed by pressing one of the five buttons on the cruise control interface: on, off, set, resume, and cancel. The bottom of *on* drives the system to the ready state *Standby*. Then by pressing *set*, the vehicle starts with the cruise control mode to follow the leading vehicle. The continuous control can be governed by a trajectory following control algorithm. Note that any brake behavior conducted by the driver will pause the cruise control mode. Then we can either cancel, turn off, or resume the cruise mode. From any non-initial state, it is possible to go back to the initial state by turning off the cruise control.

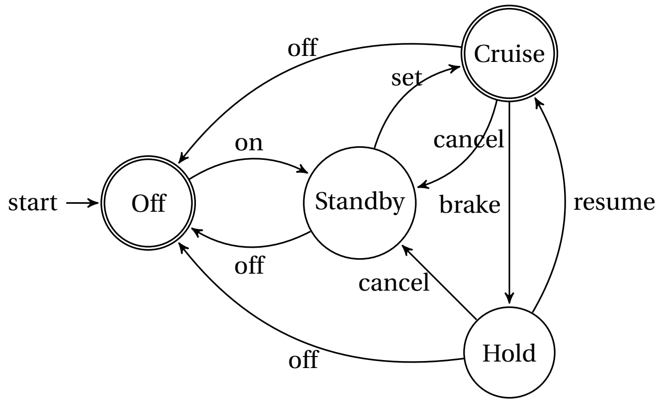


Figure 2.2: A deterministic finite state automaton models a simplified cruise controller. This example is adopted and revised from (Aström and Murray, 2010) by adding the transition from *Hold* to *Standby*.

NON-DETERMINISTIC AUTOMATA

A DFA can be extended to a non-deterministic one-NFA by considering the non-deterministic transitions in the model. In a DFA, all valid events are included in the alphabet Σ. In addition, for any state q and a transition a of the DFA, there exists a unique next state $q' = \delta(q, a)$. These are not required for a NFA. A NFA has the following formal definition:

Definition 2.4. *(Non-deterministic finite state automaton, NFA)* A NFA is a quintuple $\mathscr{A} = \langle Q, \delta, \Sigma \cup \{\epsilon\}, Q_0, F \rangle$ where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of all possible start states, $F \subseteq Q$ is a set of final states, $\delta : \Sigma \cup \{\epsilon\} \times Q \to 2^Q$ are labeled transitions with labels coming from an *alphabet* Σ.

Note that, in each transition function δ, the next state is from the power set of Q (i.e., all possible subsets of Q, of which the size is $2^{|Q|}$). In addition, the state transition is a feasible event for an empty event ϵ. The non-determinism occurs normally in two situations: first, a non-empty event drives the system in a given state to multiple states; second, the state transition is triggered by an ϵ. In a control system, for example, this situation refers to the occurring an unmodeled or unobservable event.

Definition 2.5. A run of a NFA over a string $a_1, a_2, a_3, \cdots, a_n$ is:

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$$

where $q_0 \in Q_0$, $q_i \in \delta(\epsilon^* a_i, q_{i-1}) = q_i$ for $i \in \mathbb{N}_+$, $q_i \in Q$, and $a_i \in \Sigma \cup \{\epsilon\}$. The run of an NFA is valid when $q_n \in F$. Note that an NFA is a more compact computation than a DFA: an n-state NFA can be converted to an equivalent DFA with at most 2^n states (Sipser, 2006).

Example 2.2. A simplified cruise controller with non-deterministic transitions is illustrated in Figure 2.3. Compared with the model shown in Figure 2.2, two unobservable ϵ-transitions are included in this model. The first one happens when a sudden cut-in vehicle from the adjacent lane is not detected due to some error in the sensor. We assume that the standby and hold states are under control of the driver and the cut-in vehicle is detectable and avoidable. The other unobservable ϵ-transition is a breakdown event due to, for instance, a collision.

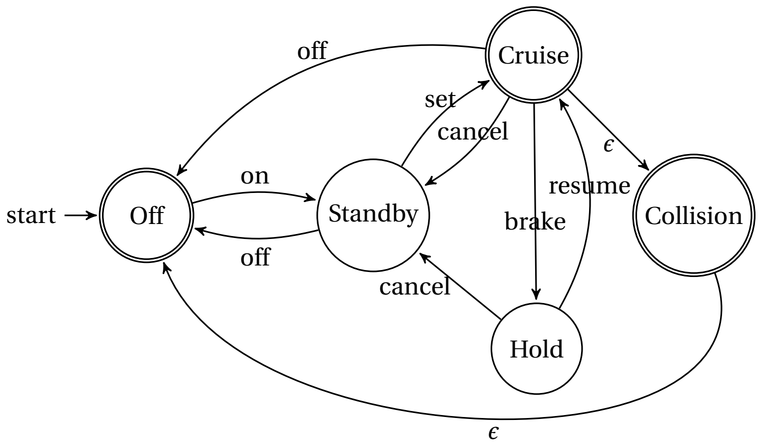


Figure 2.3: A non-deterministic finite state automaton models a simplified cruise controller.

2

2.2.3. PROBABILISTIC AUTOMATA

Definition 2.6. *(Probabilistic automaton, PA)* A PA is a quintuple $\mathscr{A} = \langle Q, \delta, \Sigma, q_0, F \rangle$ where Q is a finite set of states, $\delta : Q \times \Sigma \to p(Q)$ are labeled transitions with labels coming from an alphabet Σ and probability, $q_0 \in Q$ is the start state, $F \subseteq Q$ is a set of final states.

Definition 2.7. A run of a PA over a string $a_1, a_2, a_3, \cdots, a_n$ is a sequence of states and transitions

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$$

and its probability value $p = \prod_{i=1}^{n} \delta(q_i, q_{i+1}, a_i)$, $q_i \in Q$ and $a_i \in \Sigma$ for all $i \in \mathbb{N}_+$. The run of a PA is valid when $q_n \in F$ and $p > 0$.

Note that it is possible to assign the probability over multiple start states (initial probability) and the probability of ending a sequence in a given state (final probability). A PA can be both deterministic and non-deterministic depending on the determinism/nondeterminism of the state transition. Given a generated string and a start state, there is only one possible computation path for DPA (deterministic PA) and multiple paths for NPA (nondeterministic PA), respectively.

One of the most common ways of using probability in a PA is: in each state the probabilities of all outgoing transitions and the final probability (sequence ending in this state) sum up to one, i.e., $p_q + \sum_{q_n \in Q} \sum_{e \in E} \delta(q_p, q_n, e) = 1$, for all $q_p \in Q$, p_q is the final probability of state q. The PA models the probability distribution over all possible strings, $\sum_{w \in \Sigma^*} p(w) = 1$.

Example 2.3. A simplified cruise controller with probabilistic transitions is illustrated in Figure 2.4 as an example of a PA. In each state, the outgoing events probabilities sum up to 1. Note that, in this example, we do not model the final probability p_q in each state for simplicity's sake. For example, in the cruise state, the probabilities of brake event and turning off event are both 0.33, and the probability of undetected cut-in due to some errors is 0.01.

2.2.4. TIMED AUTOMATA

The automata described above are already powerful models for describing the logical behaviors in DES. However, the main drawback of such a representation is that the time information of events is missing. A more generic representation of sequential events in practice is using timed strings: $\tau = (a_1, t_1)(a_2, t_2) \cdots (a_n)(t_n)$, where $a_i \in \Sigma$ is an event, $t_i \in \mathbb{R}_+$ is a value, $n \in \mathbb{N}$. The time can be recorded into a *relative* form or an *absolute* form. The relative form of time t_i refers to denoting the time delay between two consecutive occurring events a_i and a_{i-1}. The absolute form of time t_i refers to denoting the exact time of a_i. A timed language is a set of timed strings over an alphabet. The corresponding computation model is called a timed automaton (TA) accepting or generating the timed language (Alur and Dill, 1994).

Note that the key additional component in a TA compared with a DFA is the clock. Generally, there are three basic operations in a clock: first, there is a function that maps a clock to a real positive value $v(x) \in \mathbb{R}_+$, where $x \in X$ is the clock; second, the clock increases or decreases over time; third, it can be reset to 0 on some conditions.

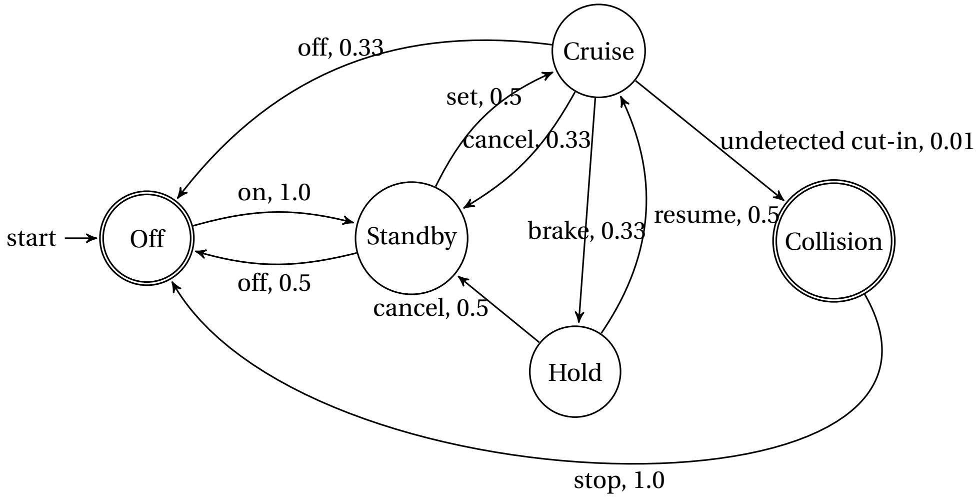


Figure 2.4: A probabilistic finite state automaton models a simplified cruise controller. The probability of the sequence: Off-Standby-Cruise-Hold-Cancel-Off is $1.0 \times 0.5 \times 0.33 \times 0.5 \times 0.5 \approx 0.04$.

Definition 2.8. A timed automaton is a 6-tuple $\mathscr{A} = \langle Q, C, \Sigma, \Delta, q_0, F \rangle$ where Q is a finite set of states, C is a finite set of clocks, Σ is the finite set of symbols, $\Delta : Q \times \Sigma \times B(C) \times 2^C \times Q$ is a set of transitions. $B(C)$ is the set of boolean clock constraints involving clocks from C. A transition $\delta \in \Delta$ is a tuple $\langle q, q', a, g, R \rangle$, where $q, q' \subseteq Q$ are the source and target states, $a \in \Sigma$ is a symbol, g is a clock guard, and $R \subseteq C$ is the set of clock resets. $q_0 \in Q$ is the start state, $F \subseteq Q$ is a set of final states.

Definition 2.9. A run of a TA over a timed string $\tau = (a_1, t_1)(a_2, t_2) \cdots (a_n, t_n)$ is:

$$q_0 \xrightarrow{a_1, t_1} q_1 \xrightarrow{a_2, t_2} q_2 \cdots q_{n-1} \xrightarrow{a_n, t_n} q_n$$

where the transition $\langle q_{i-1}, q_i, a_i, g, R_i \rangle \in \Sigma$ is valid for any $i \in n$, namely g is satisfied by the valuation v_i for all $i \in n$, $q_i \in Q$, and $a_i \in \Sigma$. The valuation v_i is defined as: $v_i(x) = 0$ if $x \in R_i$ (clock is reset), or $v_i(x) = v_{i-1}(x) + t_i$ (clock increases), and $v_0(x) = 0$, for all $x \in X$. A finite computation of a TA is called valid when $q_n \in F$.

Example 2.4. A simplified cruise controller is illustrated in Figure 2.5 as an example of a TA. In this model, there is one clock x. The goal is to control the system to recover to the *Standby* state at least 3 seconds after the brake action.

2.3. HYBRID DYNAMICAL SYSTEMS

Note that the (untimed and timed) automata models described above are used for representing the discrete behaviors of a dynamical system. To deal with both continuous and discrete dynamic behavior, a hybrid system is used to model a system that can both flow (described by a differential equation) and jump (described by a state machine or automaton).

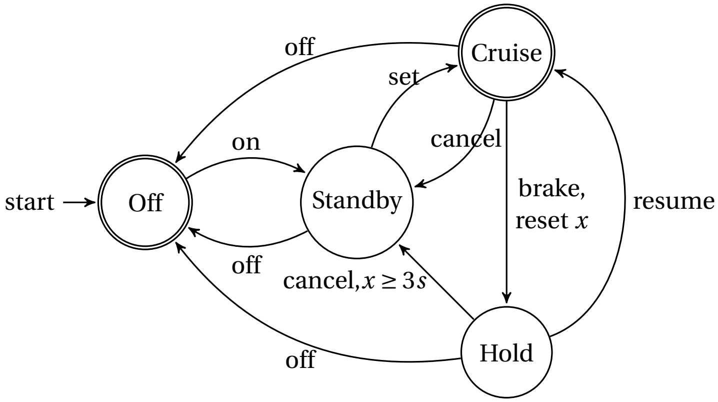


Figure 2.5: A timed deterministic finite state automaton models a simplified cruise controller. The transition from *Hold* to *Standby* relies on the additional time guard checking. The controller will stay at *Hold* when *Cancel* is executed but for no more than 3 seconds.

2.3.1. HYBRID AUTOMATA

In the following, we introduce the definition of hybrid automata (HA) using commonly used notation in the literature. To avoid possible confusions about different mathematical symbols essentially denoting the same variables, Table 2.1 shows a comparison list from HA to DFA.

Table 2.1: HA-DFA notation comparison. Note that for the initial state, HA has an extra initialization of continuous variables.

HA	DFA	Notation
Loc	Q	State
Edge	δ	State
Init(l)	q_0	Initial state

Definition 2.10. A hybrid automaton H is a tuple $<$ **Loc**, **Edge**, Σ, **X**, **Init**, **Inv**, **Flow**, **Jump** $>$ where:

- **Loc** is a finite set $\{l_1, l_2, \cdots, l_m\}$ of (control) locations that represent control modes of the hybrid system (similar to discrete states in a DFA).

- Σ is a finite set of events.

- **Edge** $\subseteq$ **Loc** $\times\, \Sigma \times$ **Loc** is a finite set of labeled edges that represent discrete changes of control modes in the hybrid system. Those changes are labeled by events from Σ.

- **X** is a finite set $\{x_1, x_2, \cdots, x_n\}$ of n-dimension real-valued variables. $\dot{X}$ is for the first-oder differential of variables $\{\dot{x}_1, \dot{x}_2, \cdots, \dot{x}_m\}$ inside a location. The primed variables $\{x_1', x_2', \cdots, x_n'\}$ are used to represent updates of variables from one control mode to another. This is called an assignment.

- **Init(l)** is a predicate for the valuation of free variables from X when the hybrid system starts from location l.

- **Inv(l)** is a predicate whose free variables are from X and which constrains the possible valuations for those variables when the hybrid system is in location l.

- **Flow(l)** is a predicate whose free variables are from $X \cup \dot{X}$ stating a continuous evolution, which is a differential equation (usually ordinary differential equation, ODE), when the control mode is in location l.

- **Jump** is a function that assigns to each labeled edge a predicate whose free variables are from $X \cup \dot{X}$. Jump(e) states when the discrete change modeled by the event e is possible and what the possible updates of the variables are when the hybrid system makes the discrete change.

Note that a TA can be represented by a HA by defining the clock's increasing or decreasing in the (flow(l)), and the reset of clock in the assignment. However, in this thesis, we would like to still consider a TA as a different model instead of a special case of a HA.

Example 2.5. A simplified hybrid cruise controller is illustrated in Figure 2.6 as an example of a HA. In this model, the location of *Cruise* comprises a continuous feedback control and an invariant, which is the valid working condition (detectable range for the equipped radar) of *cruise*. In case there is no outgoing transition and the invariant is satisfied, the system "stays" in *cruise*. The goal is to control the system to recover to the *Standby* state at least 3 seconds after the brake action.

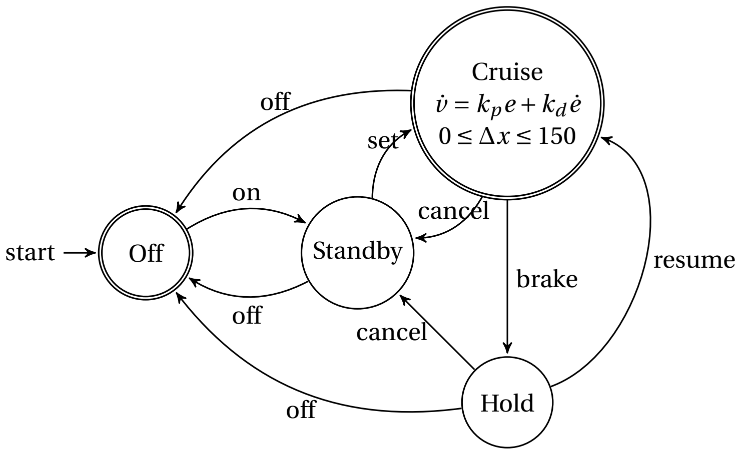


Figure 2.6: A hybrid automaton models a simplified cruise controller. The *cruise* location is governed by a proportional differential (PD) control law, where $e = \Delta x - d_{des}$, $d_{des} = d_{safe} + v$. v is the speed of the following vehicle, d_{safe} is the parameterized safety distance, Δx is the relative distance between the following vehicle and the leading vehicle. To simplify the illustration, the trivial kinetic dynamic such as $\dot{x} = v$ is not shown in the mode.

2.4. AUTOMATA LEARNING

The inference of *regular language* represented by means of finite automata is widely studied in the field of machine learning (de La Higuera, 2005). The motivation of studying this problem is because of its position in the Chomsky hierarchy. The regular language family is the simplest and best known. It can be used as the starting point to study larger families. At the same time the learning techniques developed for this problem can be extended to other domains. On the other hand, some tasks studying the dynamical systems can be dealt with on the basis of automata models. In Kin Sun Fu's work (Fu, 1977), the philosophy is that the basic organizing principle of the world is grammatical, namely composing a relatively much smaller set of words using grammar rules. The goal is to find a hierarchical or structural explanation, which is different from the flat representation used in statistical pattern recognition. Another advantage of automata learning is its unified framework integrating representation (formal language), learning (grammatical inference), and computation (on the basis of the automaton as a computation model) (Zhu et al., 2007; Fu, 1977).

As a pioneering work to theoretically study how difficult this problem is, Gold proved that given a finite alphabet Σ, two finite subsets of accepted and rejected strings $S, T \subseteq \Sigma^*$ and an integer k, determining if there exists a $k-$state DFA recognizes L ($S \subset L$, and $T \subset \Sigma^* - L$), is NP-complete (Gold, 1978b). In practice, the goal of automata learning is normally to find an automaton with the minimal size (e.g., in terms of number of states) among all hypotheses that best explains the input data. This is based on the *Occam's razor* principle (Rasmussen and Ghahramani, 2001), which is a common philosophy or heuristic in learning theory. Similar ideas can be found in *minimum description length* (MDL) (Grünwald, 2007) and bias-variance dilemma (Friedman et al., 2001). The first algorithm for grammatical inference dates back to 1967 (Gold, 1967), where the minimal deterministic automaton can be obtained in polynomial time when a representative enough set of data is available. As a follow-up work, Angluin proved that for a given incomplete set of data, finding the minimal DFA is NP-hard even for a target machine having only two states. The problem is that the absence of positive or negative samples (even for an arbitrarily small fixed fraction) poses the difficulty of providing enough evidence of distinguishing two states (Angluin, 1978). Although automata learning is hard in theory due to these "negative results", many techniques have emerged to make practical problems more tractable. The main techniques of learning DFA from positive and negative examples can be classified into four representative categories: non-merging, merging, heuristic merging, and merging with search algorithms.

2.4.1. LEARNING FROM POSITIVE AND NEGATIVE DATA

NON-MERGING ALGORITHMS

Trakhtenbrot and Barzdin proposed an inference algorithm considering all the strings of the language whose length is bounded by a given integer (Trakhtenbrot and Barzdin, 1973). Is has been shown that their work and Gold's work (Gold, 1967) are essentially similar but developed independently (Garcia et al., 2000). In the following, we call these two works the TB/Gold algorithm. The key difference lies in the representation of data. The main drawbacks are: 1) the algorithm is not incremental; 2) it doesn't have good generalization. Empirical experiments have shown its low recognition on testing data. This is because the algorithm generally works better in a sample having a characteristic set. The characteristic set consists of representative examples. The learning algorithm always returns the correct hypothesis with the characteristic set. The hypothesis does not change even if extra examples are added in the characteristic set. TB/Gold algorithm can be considered a Nerode-type learning approach (Hopcroft et al., 2006).

Theorem 1. Myhill–Nerode theorem: Given a language L, and a pair of strings x and y, define a distinguishing extension to be a string z such that exactly one of the two strings xz and yz belongs to L. Define a relation R_L on strings by the rule that x R_L y if there is no distinguishing extension for x and y. R_L is essentially an equivalence relation on strings, and thus it divides the set of all strings into equivalence classes.

The Myhill–Nerode theorem states that L is regular if and only if R_L has a finite number of equivalence classes, and moreover that the number of states in the smallest DFA recognizing L is equal to the number of equivalence classes in R_L. In particular, this implies that there is a unique minimal DFA with minimum number of states (Hopcroft et al., 2006).

An intuitive example is provided in the following to explain how Gold's algorithm works. The original example is adopted from (Garcia et al., 2000). We complete its learning steps in more detail.

Example 2.6. $D_+ = \{abb, bb, bba, bbb, babb\}$ and $D_- = \{\lambda, a, ba, aba, bab\}$ are positive and negative examples, respectively. The so-called state characterization matrix is in Table 2.2, where E is a suffix-complete set from the examples. For example, the suffixes of $babb$ are $\{b, bb, abb\}$. $S = S_1, S_2, \cdots, S_n$ is the state set, $S\Sigma - S$ is the one-letter extension from the state set. The labels 1 and 0 are associated with accept state and reject states, respectively. The label of undefined state is considered empty.

Table 2.2: Initial state characterization matrix. The obviously different rows λ and b are highlighted.

E		abb	bb	b	λ	ba	a	ab
S	λ	1	1		0	0	0	
$S\Sigma - S$	a		1		0	0		
	b	1	1	1		1	0	0

Two rows are said to be *obviously different* if they have obviously different labels $\{0, 1\}$ in some columns. Note that the undefined label, i.e., empty cell in the table, is not used as evidence of distinguishing two states. A state characterization matrix is called *closed* if no row belonging to $S\Sigma - S$ is obviously different from all rows in S. The matrix is not closed in Table 2.2 because the row of b is obviously different from the row of λ. The row of b is added to the state set as a "promoted" state, and its one-letter extension is added to $S\Sigma - S$ correspondingly. All elements in the matrix are filled out according to the input examples. As shown in Table 2.3, now bb is obviously differently from both λ and b.

Again, the state bb and its one-letter extensions are added to the matrix, see Table 2.4. The matrix is closed and ready for constructing a DFA because we can not find any row in $S\Sigma - S$ that is obviously different from all rows in S.

Table 2.3: 2nd state characterization matrix. b is promoted into S.

E		abb	bb	b	λ	ba	a	ab
S	λ	1	1		0	0	0	
	b	1	1	1		1	0	0
$S\Sigma - S$	a		1		0	0		
	ba		1	0	0			
	bb			1	1		1	

Table 2.4: 3rd state characterization matrix. Three different clusters of rows are highlighted with three different colors.

E		abb	bb	b	λ	ba	a	ab
S	λ	1	1		0	0	0	
	b	1	1	1		1	0	0
	bb			1	1		1	
$S\Sigma - S$	a		1		0	0		
	ba		1	0	0			
	bba				1			
	bbb				1			

First, from S we already know the number of states. Three distinct states of a DFA are constructed as shown in Figure 2.7. Second, we assign $S\Sigma - S$ into S according to the rows' similarities. Here we get three clusters: $\{\lambda, a, ba\}$, $\{b, bba, bbb\}$, and $\{bb\}$, which gives us the information about the reachable state. For example, λ is state 0. a and ba will also go to state 0. The three states are displayed with three different colors in Table 2.4. The finite number of equivalence also explains why the learning algorithm is called Myhill-Nerode type approach. Last, the resulting DFA is shown in Figure 2.8.

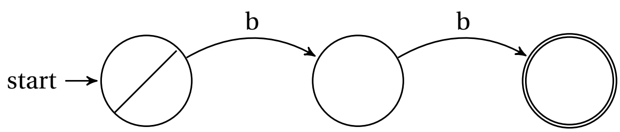


Figure 2.7: A deterministic finite state automaton learned using TB/Gold algorithm-intermediate model of construction.

Note that a non-deterministic behavior is possible in two places: 1, there could be multiple rows from $S\Sigma - S$ that are possible to add to S; 2, some transitions are possibly assigned into different states, e.g., the row of bba are compatible with rows b and bb, thus the ambiguous reachable state of bba would be state b or state bb. These two problems can be avoided by assigning the equivalence to the state with the lowest lexicographic order, i.e., in our example, bba and bbb are equivalent with b instead of bb. Unfortunately, the algorithm does not guarantee consistency with the input data. For example, bba and bbb should be accepted, while bab should be rejected, which is not

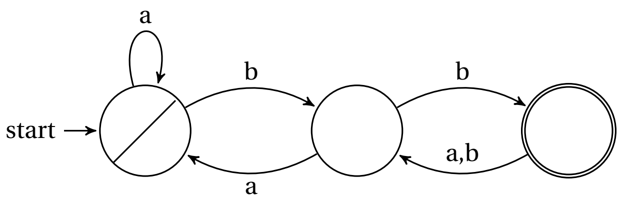


Figure 2.8: A deterministic finite state automaton learned using TB/Gold algorithm-final model of construction.

the case by looking at the DFA in Figure 2.8.

MERGING ALGORITHMS

To deal with the several drawbacks of the TB/Gold algorithm, RPNI (Oncina and Garcia, 1992) and Traxbar (Lang, 1992) were proposed in the 1990s. The main development with respect to TB/Gold was on the *state-merging* of indistinguishable states. Once one of these merges has been carried out, the algorithm keeps this current hypothesis and discards the previous one before merging. In contrast, the TB/G algorithm does not update the states during the learning procedure.

The idea of a state-merging algorithm is to first construct a tree-shaped DFA $\mathscr{A}$ called *augmented prefix tree acceptor* (APTA) from the training data, and then to merge the compatible states of $\mathscr{A}$. An APTA is a precise encoding of the input data without any generalization. It is called augmented because it contains the states that are neither accepting nor rejecting, i.e., the undefined states in the TB/Gold algorithm example. Merging the states of this APTA is essentially a learning or generalization approach that aims to find a DFA that is as small as possible. The underlying philosophy is the Occam's Razor principle (Blumer et al., 1987), i.e., a simpler hypothesis is more likely to be correct than complex ones. The APTA of the example $D_+ = \{abb, bb, bba, bbb, babb\}$ and $D_- = \{\lambda, a, ba, aba, bab\}$ used before is shown in Figure 2.9. Note that normally in APTA the states are ordered lexicographically.

A merge (see Algorithm 1) of two states q and q' combines the states into one: it creates a new state q'' that inherits the incoming and outgoing transitions of both q and q'. Such a merge is only allowed if these two states are consistent, i.e., it is not the case that q is accepting while q' is rejecting or vice versa. When a merge introduces a non-deterministic choice, i.e., q'' is the source of two outgoing transitions with the same symbol, the target states of these transitions are merged as well. This is called the determinization process, and the merge in Algorithm 1 is called detmerge. The process continues until there are no non-deterministic choices left.

The merge order depends on the state number in RPNI. For example, because state 0 does not have his father state, we start the merge from state 1 with state 0. The resulting automaton is shown in Figure 2.10. The new merged state is $q'' = \{0, 1\}$.

In Figure 2.10, the state $\{0, 1\}$ has a non-deterministic outgoing transition because the identical symbol b leads to two different target states 2 and 3. The algorithm then merges states 2 and 3. The process continues to merge state pairs 4 – 6 and 5 – 7 for the same purpose of determinization. The result of a merge is a new DFA that is smaller

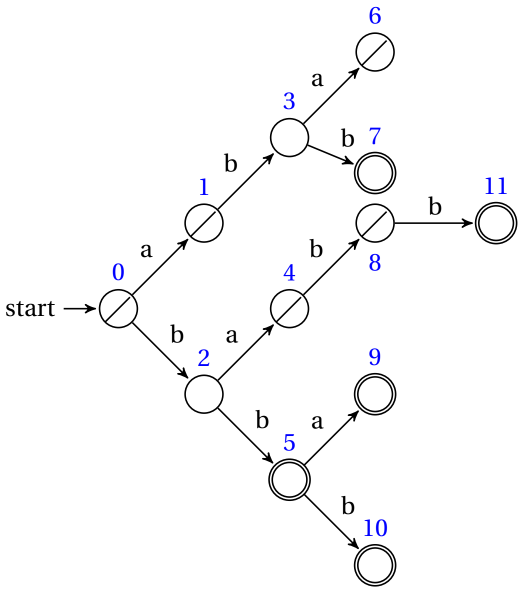


Figure 2.9: APTA of the input data. The order number of each state is highlighted with blue color.

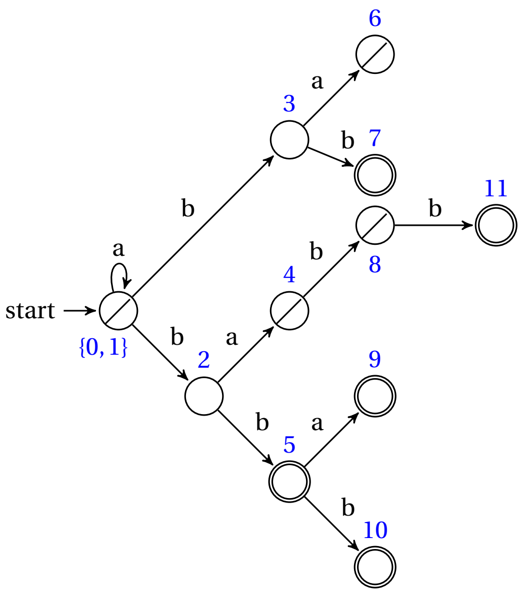


Figure 2.10: Resulting DFA after merging the states 0 and 1. We use a curly brace to represent a block merging multiple states.

Algorithm 1 Deterministic merge of states detmerge $(\mathscr{A},\ q,\ q')$

Require: an DFA $\mathscr{A} = \langle Q, T, \Sigma, q_0, F\rangle$, two states $q, q' \in Q$
Ensure: if q and q' are inconsistent, return FALSE; else return $\mathscr{A}$ with q and q' merged.
  if p is accepting state and q is rejecting state or vice versa **then**
    return FALSE
  else
    create a new state q'', and set $Q := Q \cup q''$ ▷ for consistent two states, add a new state q'' to $\mathscr{A}$
    if q or q' is an accepting (or rejecting) state **then**
      set q'' as an accepting (or rejecting) state
    end if
    for all symbols $l \in \Sigma$ **do**
      set $T(q'', l) := T(q, l)$, set $T(q'', l) := T(q', l)$ ▷ copy outgoing transitions from q and q'
    end for
    for all states $q_s \in Q$ and symbols $l \in \Sigma$ such that $T(q_s, l) \in \{q, q'\}$ **do** ▷ for all source states of transitions to q or q'
      set $T(q_s, l) := q''$ ▷ copy incoming transitions to q or q'
    end for
    for all non-deterministic choice of transition with target states q_n and q'_n **do**
      b=merge($\mathscr{A}$,q_n,q'_n)
      if $b =$ FALSE **then**
        return FALSE and undo the merge ▷ when the targets are inconsistent
      end if
    end for
    return $\mathscr{A}$
  end if

than before, and still consistent with the input sample S. The resulting DFA in this step is shown in Figure 2.11.

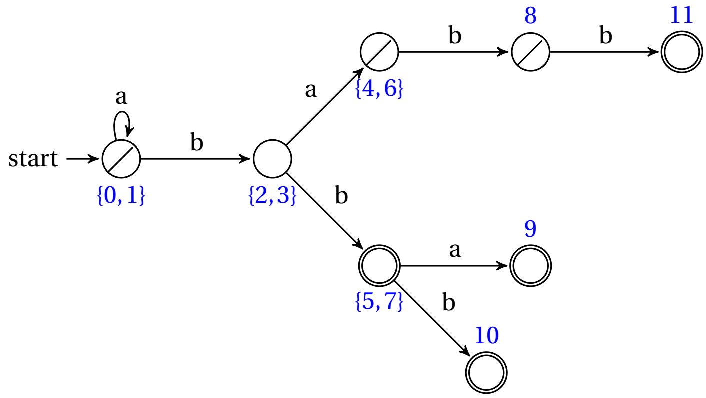


Figure 2.11: Resulting DFA after merging the states $2-3$, $4-6$, and $5-7$.

A state-merging algorithm continually applies the state-merging process until no more consistent merges are possible. Till now, we have an automaton (see Figure 2.11) having the states $\{0,1\},\{2,3\},\{4,6\},\{5,7\},8,9,10,11$, because we have conducted a state merge for the state 1. Now it is the turn for the state 2 in the original APTA ($\{2,3\}$ in the current model) according to the lexicographical order. Any block containing multiple merged states is numbered and called by the state with the smallest number within itself. For example, $\{0,1\}$, $\{2,3\}$,and $\{4,6\}$ are sorted by states 0, 2, and 4. Any block of state is merged with its precedent blocks. If multiple blocks are possible, we again consider the precedent block with the lowest lexicographical order. Unfortunately, the states $\{2,3\}$ and $\{0,1\}$ are not mergible because the resulting automaton will try to merge the non-deterministic states $\{0,1,2,3\}$ and $\{5,7\}$ due to the two outgoing transitions b from an identical source state, but $\{0,1,2,3\}$ and $\{5,7\}$ are inconsistent because they have different labels as rejecting and accepting. We continue to consider the state $\{4,6\}$ with its precedent state $\{0,1\}$ with the lowest lexicographical order. The resulting automaton is shown in Figure 2.12.

A detmerge is conducted for merging the state pairs $\{2,3\}-8$ and $\{5,7\}-11$ for determinization. Note that the newly merged state $\{2,3,8\}$ has a new rejecting label because we are trying to merge a rejecting state and an undefined state. The resulting DFA in this step is in Figure 2.13.

Because the state $\{5,7,11\}$ is incompatible with its precedent states, we merge the states $9-\{5,7,11\}$ and then $10-\{5,7,9,11\}$. The final automaton is in Figure 2.14. We can see that the DFA accepts all strings in the positive example $D_+ = \{abb, bb, bba, bbb, babb\}$ and rejects all strings in the negative example $D_- = \{\lambda, a, ba, aba, bab\}$.

HEURISTIC MERGING ALGORITHMS

Conventional state merge algorithms such as RPNI and Traxbar work well when the training set is sufficiently representative of the language. However, some merges with

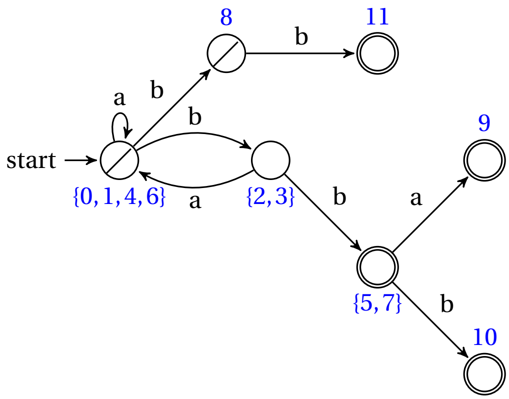


Figure 2.12: Resulting DFA after merging the states {4,6} and {0,1}.

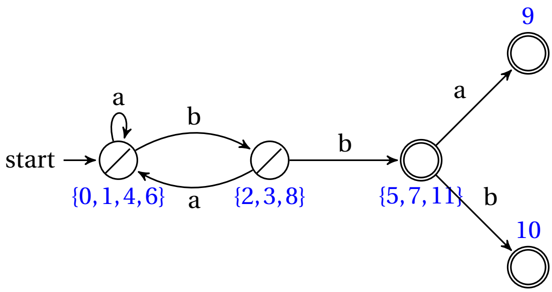


Figure 2.13: Resulting DFA after merging the states {2,3} – 8 and {5,7} – 11.

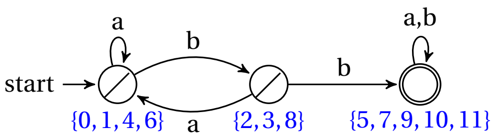


Figure 2.14: Resulting DFA after merging the states {5,7,11} – 9 and {5,7,9,11} – 10.

low evidence could be performed because of the absence of positive and negative examples in the training data. This has a negative effect on automata learning and leads to poor generalization. In the mid-1990s de la Higuera et al. firstly proposed a heuristic guided state merge algorithm to avoid the aforementioned *inconvenient* merging problem (De La Higuera et al., 1996). The state merge in this algorithm is not restricted by the lexicographical order of states anymore, which is the basic merge ordering logic of RPNI. Pairs of states of high evidence of equivalence are merged.

The approach was further developed by Price in the late 1990s and formally named as the EDSM (Evidence-Driven State Merging) algorithm (Lang et al., 1998). This algorithm achieved great success in the Abbadingo contest. The EDSM strategy is briefly summarized as follows: As in RPNI, two states p and q are compatible, where states p and q are compatible if and only if their labels are consistent. Every pair of mergible states p and q is evaluated by taking into account the number of coincidences of states with the defined output. For example, for a merge and its potential deterministic merge process, we can compute how many states with the same labels (identically accepting or rejecting) are merged. A higher score implies a better quality of merge. Once all the mergible pairs of states are evaluated, the algorithm greedily merges the pair of states with the highest score. Recalling the example of RPNI, the algorithm considers one possible merge only based on the predefined lexicographical order without evaluating multiple possible merges ordered by the score. The algorithm ends when all mergible states have been considered. The drawback of the EDSM strategy is the cost of evaluating all possible merges. The first potential improvement was also proposed in the same work (Lang et al., 1998) only considering those pairs of states at a given distance (depth) W from the initial state, which is called W-EDSM.

A further improved strategy for selecting the pairs of states to merge is the *Blue-Fringe* method (Lang et al., 1998) which is described in Algorithm 5. This algorithm is considered to be state-of-the-art with respect to the inference of DFA by merging of states. First, the algorithm initializes the red set using the initial state of the machine. The blue set is obtained by taking into account the red set, which contains those non-red states of the hypothesis that are reachable from any state in the red set. As shown in Figure 2.15, there are two possibly mergible red-blue states 0-1 and 0-2.

The algorithm ends when the blue set is empty. In each iteration, the algorithm searches for a blue state that is non-mergible with any red state. The first of such states detected is promoted to the red set and the blue set is recalculated. Any state in the blue set that is not mergible should be promoted in the hypothesis and colored as red in order to maintain consistency with the training data. If there exist blue states mergible with red states, the algorithm merges the pair of states with the greatest evidence of compatibility. It is worth noting here that the RPNI algorithm can be considered to be a Blue-Fringe method. In fact, note that if lexicographical order is considered (which is the usual order considered in the Blue-Fringe implementations) and the score computation is not carried out in the algorithm, then the algorithms do not differ from each other. Intuitively, the guided merging leads to more efficient use of the available data.

STATE MERGING WITH SEARCH ALGORITHMS

The standard EDSM algorithm is essentially a greedy program, i.e., in each iteration, only one merge with the highest score is performed. An alternative approach is to con-

Algorithm 2 State merge in the Blue-Fringe

Require: an input sample S,
Ensure: $\mathscr{A}$ is the smallest DFA that is consistent with S

$\mathscr{A} = \mathsf{APTA}(S)$ ▷ construct the prefix tree
$R = \{q_0\}$ ▷ color the start state red
$B = \{q \in Q \setminus R \mid \exists l \in \Sigma : T(q_0, l) = q\}$ ▷ color all its children blue
while $B \neq \emptyset$ **do** ▷ while $\mathscr{A}$ contains blue states
  if $\exists b \in B$ s.t. $\forall r \in R$ holds $merge(\mathscr{A}, r, b, t_d) = \text{FALSE}$ **then** ▷ if a blue state is inconsistent with all red states
    $R := R \cup \{b\}$ ▷ color b red
    $B := B \cup \{q \in Q \setminus R \mid \exists l \in \Sigma : T(q, l) = q\}$ ▷ color all its children blue
  else
    if for $b \in B$ and $r \in R$ merge$(\mathscr{A}, r, b) == True$ **then**
      call the merge$(\mathscr{A}, r, b)$ ▷ perform the merge
    else Change the color of a blue state in to red state
    end if
    Change the color of all uncolored children of red states to blue
  end if
end while
return $\mathscr{A}$

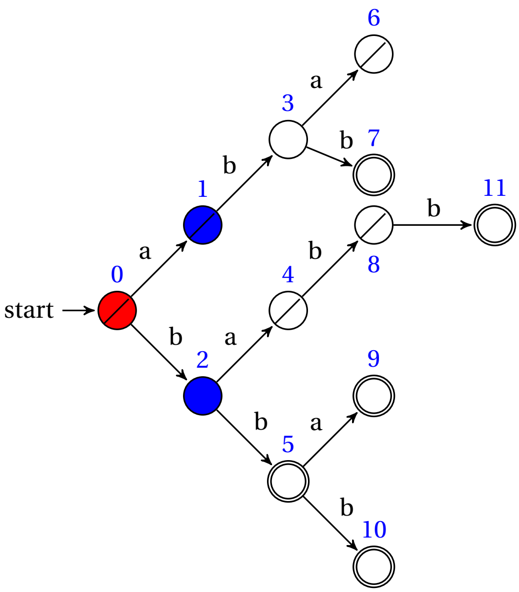


Figure 2.15: APTA in Blue-Fringe. Red nodes, blue nodes, and white nodes represent identified states, candidate mergible states, and pending states, respectively. The blue states are essentially the children nodes of the red states.

sider state merging as a sequential decision-making process of search for a smallest (or as small as possible) automaton. There are two categories of search-based state merging algorithms:

Exact algorithms: (find the smallest consistent DFA) HMM (Oliveira and Edwards, 1996), BICA (Oliveira and Silva, 2001), EXBAR (Lang, 1999).

Approximate algorithms: (find a small but not necessarily minimum-size consistent DFA, find an approximation by wrapping backtrack search around EDSM) ED-BTS (Bugalho and Oliveira, 2005), SAGE (Juillé and Pollack, 1998), ED-BEAM (Lang, 1999), ED-SS (Bugalho and Oliveira, 2005).

2.4.2. LEARNING FROM POSITIVE EXAMPLE

It has been proven that identification in the limit from only positive examples is undecidable (Gold, 1967). In a lot of cases in practice, we only have positive examples from the normal behaviors of a system. The negative examples are expensive or even impossible to obtain. The problem is formalized as learning a probabilistic automaton (PA) representing the distribution over strings.

The main difference between DFA and PA state-merging algorithms is the check for compatibility. In DFA state-merging they are compatible if there is no inconsistency. In PA state-merging they are compatible if some statistical criterion is satisfied. The most representative algorithm ALERGIA uses a compatibility measure derived from the Hoeffing bound (Carrasco and Oncina, 1994). Using this criterion two states q and q' are α-compatible if the following two conditions hold for all $e \in \Sigma$:

1. $\left|\frac{f_q}{n_q} - \frac{f_{q'}}{n_{q'}}\right| < \sqrt{\frac{1}{2}\ln\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n_q}} + \frac{1}{\sqrt{n_{q'}}}\right)$
2. $\delta(q,e)$ and $\delta(q',e)$ are μ-compatible

The first condition defines the compatibility using a precision parameter α. n_q and $n_{q'}$ are the number of strings arriving (including passing and ending) in the states. f_q and $f_{q'}$ are the number of strings ending or following a transition in the states q and q'. In other words, this condition first checks two states' compatibility by looking at their ending frequencies; then checks the compatibility for each pair of outgoing transitions. The second condition requires that the compatibility is satisfied in every pair of children of q and q'. Another difference is in the stopping condition of the merging algorithms. A DFA state-merging algorithm stops when all possible merges are inconsistent. A PA merging algorithm can have a statistical stopping criterion. The ALERGIA algorithm stops when all possible merges are not α-compatible. It can be shown that the ALERGIA algorithm identifies PAs in the limit with probability one (De La Higuera and Thollard, 2000).

We use the positive data as an example: $D_+ = \{10abb, 20bb, 30bba, 40bbb, 50babb\}$. The number associated with every string is the frequency. We first build a probabilistic APTA as shown in Figure 2.16. The frequencies of arriving and ending are displayed beside the states. The transition and its frequency are beside every arc. For example, now we are trying to merge the states 1-2, and the threshold α is arbitrarily set to 0.8. First, we check these two states' compatibility: $|0-0| < \sqrt{\frac{1}{2}\ln\frac{2}{0.8}}\left(\frac{1}{\sqrt{10}} + \frac{1}{\sqrt{140}}\right) \approx 0.27$. Then, we continue to check their outgoing transition a: $|0 - \frac{50}{140}| = 0.36 > 0.27$; b:

$|\frac{10}{10} - \frac{90}{140}| = 0.36 > 0.27$. These two transitions are both not compatible. Therefore, we conclude that the states 1-2 are not mergible. The algorithm continues to search for other mergible states, which is skipped here for compactness.

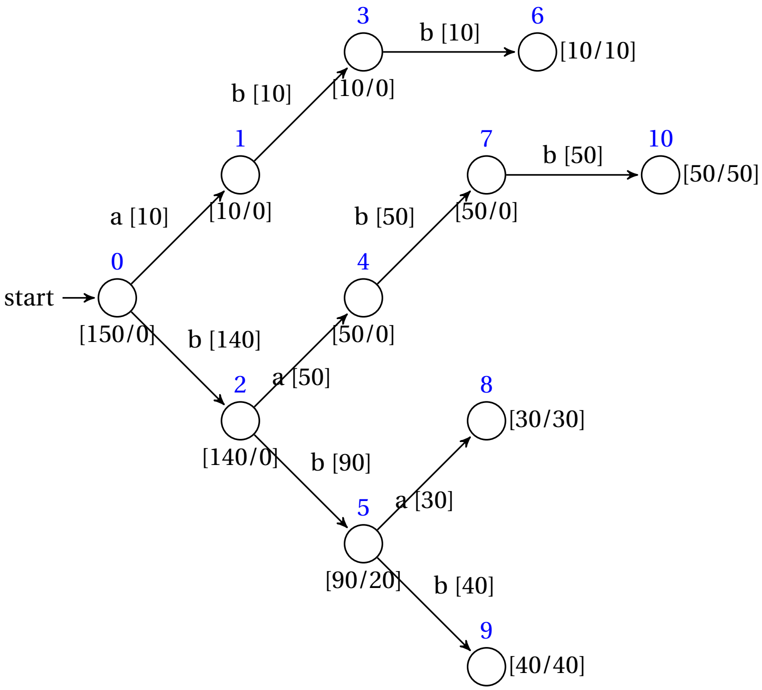


Figure 2.16: Probabilistic APTA of the positive input data. The negative example is no longer available for the construction.

2.4.3. HYBRID AUTOMATA LEARNING

The problem of hybrid dynamical systems learning is studied by both the control and the machine learning communities. The most fruitful outcome in the control domain (particularly the sub-domain as *system identification*) is the study of switched, piece-wise affine (PWA) models using Algebraic, Clustering, Bayesian, Bounded-Error optimization techniques (Paoletti et al., 2007). The general idea is to identify a model minimizing the within-domain error. The domains are based on the space partition of the variables, and they are normally mutually exclusive, i.e., there is no overlap among different domains. These approaches only deal with a piece-wise linear model where the current state is a linear combination of previous states and input. The identification algorithms either assume the order (how many steps the historical data rely on) or the number of states *a priori*.

A similar problem is formulated in the machine learning domain as multi-modal learning. Here we only review *multi-modal input-output models* to deal with a control problem. The main idea about a multi-modal model is that a complex process is formed by different modalities, which are characterized by different statistical properties. A hybrid model is essentially a multi-modal model, where each modality is governed by for instance a continuous dynamical model. Several works on multi-modal learning are dedicated to obtaining a model insightfully represented by a finite state automaton (Omlin and Giles, 1996; Hou and Zhou, 2018). These works are a type of compromise at the abstraction level instead of direct learning. We summarize two categories as *Stochastic model* and *Hybrid automaton*.

- Stochastic model:

  The input-output HMM (IOHMM) is essentially a probabilistic finite state automaton and closer to a continuous Hidden Markov Model (Bengio and Frasconi, 1995). IOHMM allows for input and output vectors, and it retains the probabilistic feature like HMM. In IOMHMM, the hidden states are assumed to follow a multinomial distribution that depends on the input sequence, which also means the dynamic of state transition is governed by the transition probability and the input control signal. The observation, i.e., the output signal vector relies on a Gaussian density with parameters depending on the current hidden state. An expectation maximization (EM) algorithm can be used for finding the optimal parameters of the model. EM iteratively rotates between an expectation (E) step and a maximization (M) step, The E step creates an expectation function of the likelihood using the current estimated parameters. The M step computes parameters maximizing the expected likelihood found in the E step.

- Hybrid Automaton:

  Instead of modeling the dynamics in a stochastic manner, the hybrid automaton aims to describe the continuous dynamic in each modality using a continuous formula, according to the linear and non-linear feature in each modality (namely a location in a hybrid automaton).

  Multi-modal symbolic regression (MMSR) is proposed to learn a non-linear formula in the locations and the transitions. MMSR consists of two sub-algorithms: the cluster symbolic regress (CSR) and transition modeling (TM). CSR is based on a combination of symbolic regression (SR) and expectation-maximization (EM). SR is built on a genetic algorithm searching for a suitable mathematical formula "best" explaining the training data both in terms of accuracy and simplicity. SR does not assume any shape of formula as *a prior*; the initial and new expressions are formed by combining mathematical building blocks like mathematical operators such as $+, -, \times, \div, etc$. Simultaneously, the EM algorithm serves as an optimization component to identify the modal data membership. TM is an algorithm to infer symbolic inequalities for binary classification boundaries for the transition conditions. Again, TM is built on the basis of SR by searching classification expressions. The benefit of MMSR is that complex non-linear formulas can be built up in each location.

A common drawback of the aforementioned hybrid systems learning is the need for fixing the number of states in advance. Automata learning can help with this problem because the number of states is identified by the learning algorithm. A *composed type of learning* for hybrid automata is proposed in (Niggemann et al., 2012). The original time series data can be represented as sequences of tuples consisting of discrete and numeric events:

$$(\mathbf{e_1}, \mathbf{v_1}), (\mathbf{e_2}, \mathbf{v_2}), (\mathbf{e_3}, \mathbf{v_3}), \cdots, (\mathbf{e_n}, \mathbf{v_n}) \tag{2.1}$$

The idea of the composed type of learning is quite straightforward. First, the discrete data are used for learning the conventional DFA. Note again the hybrid automaton we discussed here is only from the positive examples. Second, we aggregate the numeric data in each state. To achieve this, we need to keep track of the original training data in tuples, the discrete data entering into specified states implies the corresponding numeric data should also be put into such states. Third, the regression model purely for the time-driven dynamic is able to be identified from the numeric data in each state. The representative work in this category is HyBUTLA (Niggemann et al., 2012). In this work, the discrete events are obtained based on actuators' state change. They use ALERGIA-like statistical testing for the state merging in a bottom-up way. The authors claim that the bottom-up strategy merging from leaf nodes is more efficient than the conventional top-down strategy like the Blue fringe framework. The continuous behaviors in modes are identified by linear regression or a feedback neural network. The main drawback of doing so is the high complexity of the model. The number of distinct regression models is equivalent to the number of states in the finite automaton.

2.5. HYBRID SYSTEM VERIFICATION

The verification problem in control is that of considering a controller that has already been designed and connected to its plant and the environment, which is subject to some disturbances, we need to verify that all the behaviors of the system stay within a desired range of operation and do not reach a forbidden state. Note that in this thesis, the controller is not designed but learned from data. But the problem is similar to verifying that the learned behavior of the system is desired. The question can be answered by first computing the reachable set of the system subject to uncontrolled interaction with the external environment, then checking if all reachable states satisfy the property, e.g., safety studied in this thesis.

2.5.1. REACHABILITY FOR HYBRID DYNAMICS

We refer the readers to a detailed introduction of verifying continuous and hybrid systems (Maler, 2014). The history of this topic can be found in (Alur, 2011). Here we only go through the fundamental definitions and basic algorithms of reachability analysis.

Consider a continuous dynamical system $\dot{x} = f(x, v)$, where $x \in X$ is the state variable and $v \in V$ is the admissible input variable. That is to say, such a system is subject to external disturbances modeled by v. Computing the reachable set (given the initial state set $X_0 \in X$, all possible trajectories of states visited) allows one to verify that all the behaviors of the system stay within a desired range of operation and do not reach a forbidden region of the state space. Proving such properties for systems subject to uncontrolled interaction with the external environment is the main issue in verification. Note that the external disturbances are modeled by a set of admissible inputs, which is not the case for a general control problem where the disturbances are modeled by some specified probabilistic distributions. Normally, for the verification problem, we only know the ranges or bounds of the input signals.

Indeed, *numerical simulation* also deals with such a validation problem by repeatedly picking one distinct initial condition and one input stimulus producing the corresponding trajectory and observing whether this trajectory behaves properly. The obvious drawback is that all possible trajectories are unenumerable. Reachability analysis achieves the same goal by exhaustively exploring the state space in a search manner, e.g., breadth-first. We compute at each time step all the states reachable by all possible one-step inputs from states reachable in the previous step. Though its computation is much more costly than the simulation of an individual trajectory, it provides more confidence and guarantees about the correctness of the system than the limited number of numerical simulations.

A trajectory is a measurable sequence (signal) defined by a partial function $\xi : T \to X$ over all T (an infinite trajectory) or over an interval $[0, t] \subset T$ (a finite trajectory), wherein $T = \mathbb{R}_+$ is a time domain and $X \subseteq \mathbb{R}^n$ is a state space. We use the notation $T(X)$ for all such trajectories and $|\xi| = t$ to denote the signals' duration. We use $\mathscr{T}(V)$ to denote input signals $\zeta : T \to V$, where $V \subseteq R^m$ is the input space. A continuous dynamical system $S = (X, V, f)$ can also be defined as $\dot{x} = f(x, v)$.

ξ is the response of f to ζ from x if ξ is the solution of the differential equation for initial condition x, i.e., $\xi = f_x(\zeta)$ or $x \xrightarrow{\zeta/\xi} x'$. x' is said to be reachable from x by ζ within t:

$$R(x, \zeta, t) = \{x'\} \tag{2.2}$$

For all initial states represented by X_0, all time instants in an interval $I = [0, t]$, and all admissible input signals in $\mathscr{T}(V)$, the reachable set is defined as:

$$R_I(X_0) = \bigcup_{x \in X_0} \bigcup_{t \in I} \bigcup_{\zeta \in \mathscr{T}(V)} R(x, \zeta, t) \tag{2.3}$$

Figure 2.17 is a sketch illustrating the trajectories from many runs of simulation and the reachable set from the initial state set X_0 with all possible inputs. The reachable set consists of all possible trajectories within the time interval I.

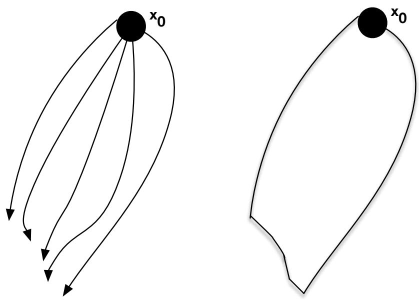


Figure 2.17: Trajectories of simulation and reachable set.

The reachability of the discrete or continuous dynamics can be computed incrementally as:

$$R_{[0,t_1+t_2]}(X_0) = R_{[0,t_2]}(R_{[0,t_1]}(X_0)) \tag{2.4}$$

The reachable states for every $R_{[0,t_i]}$ are explored by a reachability algorithm shown in Algorithm 3. Note that the total time length is L, which is chunked equally as L/r intervals. In every interval r, we compute the newly explored set P. The termination condition in Algorithm 3 is a bounded horizon by L/r times of execution. For an unbounded horizon reachability exploration, the termination condition is replaced by $P \subset Q$, namely the newly computed reachable state has already been explored. This condition usually leads to undecidability.

Algorithm 3 Reachability algorithm:

Require: Initial set $X_0 \subset X$
Ensure: $Q = R_{[0,L]}(X_0)$ $P := Q := X_0$
  for all $i = 1, 2, \cdots, L/r$ **do**
    $P := R_{[0,r]}(P)$
    $Q := Q \cup P$
  end for

Equation 2.5 shows the trajectories from a simple hybrid automaton with two states, each with its own dynamics. In this example, we simply assume the dynamic is piecewise-linear or piecewise-affine. An (extended) state of a hybrid system is a pair $(l, x) \in \mathbf{Loc} \times X$ where l is the discrete location. A transition from state l_i to state l_j may occur when the condition G_{ij} (the transition guard) is satisfied by the current value of x. Such conditions are typically comparisons of state variables with thresholds or more generally linear inequalities. Moreover, while staying at the discrete state s, the value of x should satisfy additional constraints, known as state invariants.

$$(l_1, x[0]) \xrightarrow{t_1} (l_1, x[t_1]) \rightarrow (l_2, x[t_1]) \xrightarrow{t_2} (l_2, x[t_1 + t_2]) \rightarrow \cdots, \tag{2.5}$$

The basic idea of exploring the state space in this model is shown in the sketch of Figure 2.18. To illustrate in a simplified way, the dimension is only two and every newly explored state is represented using a rectangle. First, continuous reachability is applied using the dynamics A_1 of l_1, while respecting the state invariant I_1. The initial state in l_1 is in blue. Then the set of reachable states is intersected with the transition guard G_{12}. The outcome serves as an initial set of states in l_2. Note that the intersection set is actually a polygon. Because we simply represent every state using a rectangle, we get the over-approximated rectangle as the intersection set and use it as the initial state in location l_2. The continuous linear reachability with A_2 and I_2 is applied and so on.

The key challenge is how to conduct efficient implementation of the reachability algorithm, which is one of the main research lines in the hybrid verification domain. The researchers seek for a suitable representation for the set of states supporting the operations used by the reachability algorithm. HyTech was the first model checker to implement symbolic reachability analysis for hybrid systems (Henzinger et al., 1997b). The

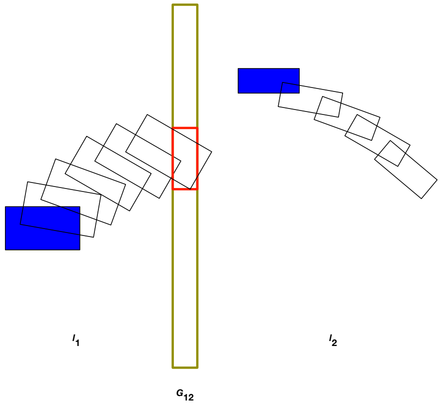


Figure 2.18: Reachable set in two states. The green box is the guard G_{12} as the transition condition from l_1 to l_2

reachable set is represented by a union of n-dimensional polyhedra, where n is the number of variables. A polyhedron is essentially a conjunction of linear inequalities over variables. However, the model is only restricted to the class of linear hybrid automata (LHA), i.e., the guards, assignments, and invariants are all linear expressions over constant constraints and some order derivatives. For example, a LHA-admissible flow is $x' = y'$ in a location and $c_1 \leq x' \leq c_2$ is a constraint in a location or an invariant, where x and y are the LHA's two variables, and c_1 and c_2 are the constants as a lower bound and an upper bound. For LHA, the polyhedral representation is closed for both discrete transitions and continuous evolution in Equation 2.5. However, unfortunately, LHA can only handle simple dynamic systems. For a more complex system, to use HyTech as a model checker, it should be over-simplified into LHA.

HyTech does not even support the most commonly used linear dynamical systems under the linear differential equation form: $\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$, where x represents the vector of state variables, $\mathbf{u}$ represents the vector of input variables, $\mathbf{A}$ is the *system matrix*, relating how the current state affects the state change $\mathbf{x}'$, $\mathbf{B}$ is the *control matrix*, determining how the system input affects the state change. The class of hybrid automata where guards, assignments, and invariants are linear expressions and the dynamics are linear differential equations, is called Linear Hybrid Systems (LHS). Even for an autonomous linear system without input, the "exact" representation is: $x = x_0 e^{\mathbf{A}t}$. This representation is not useful in practice due to its high complexity: checking the membership of a point x in this set is just solving the reachability problem itself. Here we will introduce an important concept called *flowpipe approximation* using a representative technique used in the tool d/dt. Briefly speaking, a flowpipe is a bundle of trajectories in the state space. To deal with the reachable states in a linear affine dynamical system without input, d/dt proposes to conduct the following steps. First, the states at step $k-1$ are represented by a convex hull $F^{k-1} = conv(\mathbf{V}^{k-1})$, where $\mathbf{V}^{k-1} = \{\mathbf{x_1}^{k-1}, \ldots, \mathbf{x_m}^{k-1}\}$, m is the number of vertices. Second, due to the convexity-preserving property, we compute the reachable convex hull by only using vertices $\mathbf{V}$ in a finite step δ, i.e., $G^k = conv(\mathbf{V}^{k-1} \cup \mathbf{V}^k)$. Figure

2.19 shows an example of computing the reachable state from X_0 within time δ, the blue convex hull is G^k.

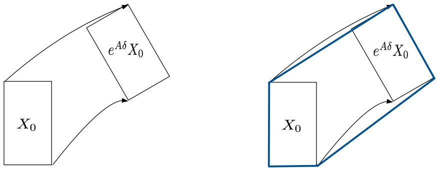


Figure 2.19: Reachable set without over-approximation. We can see that some reachable states are not included in the blue convex hull.

Note that G^k is not an over-approximation of $\delta_{[0,r]}(conv(\mathbf{V}^{k-1}))$, because the reachable states in the intermediate time between $[0,r]$ are not guaranteed to be included. Third, as shown in Figure 2.20, such a guarantee is achieved by an approximation using a "bloating" operation pushing G^k outward to get an over-approximated convex hull (polyhedron) G'^k.

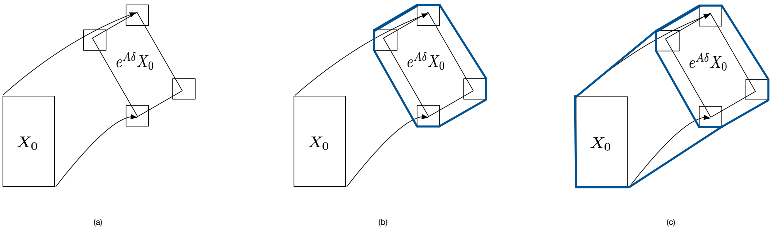


Figure 2.20: A bloating operation to guarantee that all reachable states are included.

Fourth, G'^k is further over-approximated by a "griddy" polyhedron G''^k as shown in Figure 2.21 to achieve a much less expensive representation when we check way more easily the termination condition $P_k = P_{k-1}$, where $P^k = P^{k-1} \cup G''^k$.

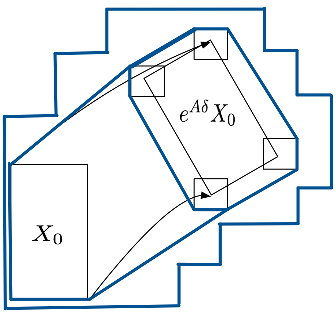


Figure 2.21: A further over-approximation by using a orthogonal polyhedron.

For the linear dynamical system with input, the state is represented as $F^{k-1} = conv$ $(\mathbf{V}^{k-1} \bigoplus \mathbf{U})$. Intuitively, the input or disturbance represented by a polytope is added up to the vertices again to form a new convex hull, as shown in Figure 2.22.

Figure 2.22: Bloating operation for input control.

We can observe that by a bloating operation in both Figure 2.20 and Figure 2.22, the resulting convex polytope will have more vertices than the original rectangle, which inevitably increases the complexity of the representation. The similar “face-lifting" technique applied as the bloating operation can be used again to guarantee the resulting polytope has the same number of vertices with the price of over-approximation error, which is shown in Figure 2.23. Such a way of keeping representation size small will accumulate errors, which is called the “wrapping effect" (Kühn, 1998).

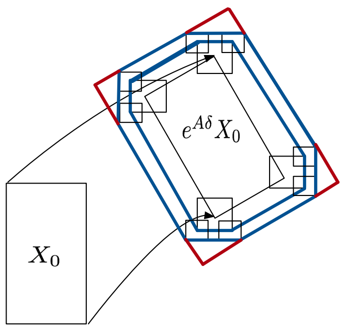


Figure 2.23: Face-lifting to keep same number of vertices

The solution is using a *lazy representation* (Girard et al., 2006). The basic idea is that the reachable state P_k can be computed from P_0, and an approximated polytope is obtained with any desired precision. In the next step, P_{k+1} is also computed from P_0 instead of from P_k to avoid accumulating errors. This technique, and more efficient representation using zonotope (Girard, 2005) and support functions, is the foundation for the state-of-the-art tool SpaceEx (Frehse et al., 2011). SpaceEx is able to handle the reachable set after 1000 steps for a 200-state variable linear system within 2 minutes.

Once the reachable set is available, we can easily check some properties of the system. The property is usually written as some logical expressions such as inequality formulae. For example, in a cruise control system, we can do the safety verification by checking if the relative distance of two cars $\Delta x > 0$ always holds in all reachable states. Or we can define a “bad state", where $\Delta x <= 0$, and check if this state is reachable using the reachability analysis, e.g., computing the intersection of the system's reachable states and the bad state.

2.6. SUMMARY

In this chapter, we first introduce several automata models using examples of cruise control systems. Some computation models such as probabilistic automata and hybrid automata will be used as the modeling tools in the coming chapters.

Second, we provide a gentle tutorial about automata learning algorithms starting with the TB/Gold algorithm, a Myhill-Nerode like approach to the state-of-the-art state merge algorithms. In the coming chapters, we will continue to introduce a more advanced regression automaton model and its learning algorithm, which is built upon conventional state merge algorithms. For one of the main research lines in this thesis, related work on hybrid model learning is presented. Our work is mostly in the category of *hybrid automata learning*. Our first algorithm to infer hybrid automata presented in Chapter 3 is closely related to HyBUTLA (Niggemann et al., 2012), using a type of *composed learning*. To overcome the high complexity problem in HyBUTLA, we propose to further abstract the states to form more high-level modes based on their sub-sequence similarity. The second algorithm presented in Chapter 4 is called *inline learning*, which considers the numerical features of the raw continuous data during the state merge procedure. We argue that this novel algorithm is more compact than *composed learning*.

Last, we briefly go through many fundamental concepts in the verification of hybrid systems such as simulation, reachable set, reachability algorithm, state representation, etc. We believe that the reachability analysis can be leveraged as a powerful tool for verifying the hybrid models we learn using our algorithms. Chapter 7 indeed showcases how to use this tool to verify a data-driven adaptive cruise controller learned from human driving data. Unfortunately, the hybrid model checkers in existence do not support the models we learn. In Chapter 7, we make a contribution by proposing a format transforming tool to fill and close this gap.

3

LEARNING HYBRID AUTOMATA FOR IMITATION CONTROL

In this chapter, a novel algorithm based on a composed learning strategy for a multiple mode hybrid automaton model (MOHA) is discussed. A discrete timed automaton model is first learned as a "skeleton" representing the logical evolution of discrete dynamics. To deal with multiple modes of dynamical behaviors consisting of multiple states, the states are abstracted into modes on the basis of their behavioral similarities. A continuous dynamical function is then used for describing continuous dynamics in each mode.

MOHA is applied to learn the car-following behavior of human drivers. The discrete timed automaton is used for modeling traffic environment evolution. The modes represent short, medium, and long distance car-following, free driving are abstracted. The continuous car-following equation in each model is used for continuous control of longitudinal acceleration/deceleration. The model is then used for the traffic simulation and the human-like car-following controller design.

The material in this chapter has appeared in

- Qin Lin, Yihuan Zhang, Sicco Verwer, and Jun Wang. Moha: a multi-mode hybrid automaton model for learning car-following behaviors. IEEE Transactions on Intelligent Transportation Systems, (99):1–8, 2018
- Yihuan Zhang, Qin Lin, Jun Wang,and Sicco Verwer. Car-following behavior model learning using timed automata. IFAC-PapersOnLine, 50(1):2353–2358, 2017
- Yihuan Zhang, Jun Wang, Qin Lin, Sicco Verwer, and John Dolan. A data-driven behavior generation algorithm in car-following scenarios. In Dynamics of Vehicles on Roads and Tracks Vol 1: Proceedings of the 25th International Symposium on Dynamics of Vehicles on Roads and Tracks (IAVSD 2017), page 227. CRC Press, 2017

3

3.1. INTRODUCTION

Car-following is the most common behavior in daily driving. Learning car-following is of great importance for a subject vehicle to monitor, estimate or even predict the states of nearby vehicles for interaction and decision-making. A car-following model essentially reflects how a driver responds to his or her existing driving states by implementing a certain action. A more formal definition is that this model tries to bridge *input stimuli* or *explanatory variables*, like subject vehicle speed, relative distance and relative speed to a leading vehicle, and *output actions* or *response variables*, like acceleration or deceleration. The first work on car-following can be dated back to the 1950s, in which models were developed to evaluate traffic capacity and congestion. A linear follow-the-leader model was proposed in Ref. (Pipes, 1953) that bridged the driver's desired acceleration and the speed difference between the following and the leading vehicles. Another widely used linear model was proposed in Ref. (Helly, 1959a). Alternatively, non-linear models in Ref. (Gazis et al., 1961) introduced power operators of range and speed. An intelligent driver model (IDM) was developed in Ref. (Treiber et al., 2000), which was a time-continuous car-following model for the simulation of freeway and urban traffic. Genetic algorithms are the most widely used techniques to identify parameters in the aforementioned models. A *gross fitting* strategy is usually used for identification, i.e., fitting a car-following model on all the collected data. The gross fitting inevitably has large fitting errors, and is more suitable for overall traffic flow simulation. Most of the existing car-following models using gross fitting do not fully capture driver behavior in different driving scenarios (Hamdar et al., 2008). Driving behavior actually includes heterogeneity of *inter-driver difference* and *intra-driver difference* (Van Hinsbergen et al., 2015). The inter-driver difference, discovering that different car-following models may apply to different drivers, is useful for driving behavior modeling and skills evaluation of individual drivers (Hoogendoorn et al., 2006), which is not the focus of this work. The intra-driver modeling basically deals with the problem that individual drivers change their behaviors over the data collection period. This chapter aims at learning a model averaging driving behaviors from thousands of human drivers from a data-driven perspective, where cognitive parameters are identified from real driving data. The basic idea is discretizing the environmental variables, i.e., speed, relative distance, and relative speed on a coarse-grained level and obtaining a stateful model. Distinct driving patterns or modes are obtained by partitioning such a model into groups of states based on states' sequential similarity. Corresponding groups of car-following models are identified on a fine-grained level from the real-value time series data. Using such a ***divide-and-conquer*** learning, the approximation error of this switching car-following model is expected to be lower. Meanwhile, the underlying dynamic of driving behavior is discovered.

This work is motivated by Ref. (Higgs and Abbas, 2015), which dealt with the similar tasks of patterns mining and divide-and-conquer learning in the car-following model. In their paper, they first segmented the time series driving data by means of *change point detection*, and afterwards mean values representing the segmented piece-wise data were clustered using the k-means algorithm. The noticeable disadvantage of this approach, formally called ***feature vector clustering*** (Smyth, 1997), is that it loses sight of dynamic and time information. In addition, the obtained clusters are not interpretable, and the switching mechanism among clusters is missing. These problems will be solved by the

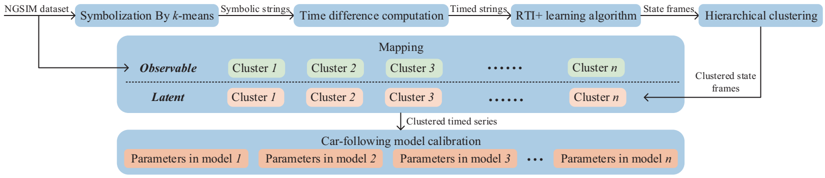


Figure 3.1: The flowchart of the proposed approach. The clustering is deployed on the state sequences (the latent variables under the dotted line in the module). The original numerical time series data are also clustered correspondingly with the help of the mapping. Different car-following models are trained from the clustered time series data, one for every cluster.

timed automaton model in this chapter. Another related work is Ref. (Verwer et al., 2011), which recognized truck driving behaviors, like accelerating too fast or normal acceleration, from labeled sequences. In this chapter, a ***sequence clustering*** is deployed based on the input data. It essentially clusters similar driving processes shared in multiple complete car-following periods. This work also focuses on obtaining interpretable models from unlabeled sequences. Instead of learning semi-supervised classifiers separately from the sensors of speed and fuel engine, as was done in Ref. (Verwer et al., 2011), the framework in this chapter is a unique generative model with distinguishable behaviors in different model regimes. Due to their insightful and interpretable properties, automata have been widely used for modeling more complex driving behaviors like lane change, intersection access, and turning, to name a few (Schwarze et al., 2013; Bouhoute et al., 2014; Gadepally et al., 2014).

The original multivariate time series data are discretized from a widely used public dataset into symbolic strings. A symbolic representation of time series data has the following benefits: it provides a high-level overview of behavioral dynamics; it significantly reduces the dimensionality of multi-variate time series data; it is robust to noise; it has inexpensive similarity computation for discovering driving patterns. Such a symbolic representation is sufficient for conventional discrete event system modeling. However, in many application settings, time information is crucial for behavior modeling. For example, moderate and harsh decelerations are obviously not the same driving behavior. The time difference between two consecutive distinct events is therefore computed to obtain timed strings. The learning process benefits from such timed sequential data since they help *explicitly* discover the underlying varying-duration behaviors. Then a state-of-the-art automata learning algorithm named RTI+ (real-time identification from positive data) is deployed to learn a direct and cyclic graphical model that "best" describes the observed data. With the help of this structural model, frequent common *state sequences* as patterns are extracted and clustered. A complete car-following period consists of distinguishable temporary behaviors represented by the aforementioned clusters. The corresponding original time series data are also clustered by mapping their indices. Car-following models are trained in the obtained individual clusters of time series data. Figure 3.1 shows a flowchart of the proposed approach.

This chapter makes the following contributions:

1. Multivariate time series data are represented with symbolic timed strings, and a highly interpretable model is learned with state-of-the-art automata learning al-

gorithms.

2. Properties of temporal processes, i.e., sequential features, are used for clustering the input data. The results show that the fitting accuracy is significantly improved.

3. To the best of our knowledge, this is the first work to use state sequence clustering to label different behaviors in an automaton by partitioning the model.

4. The usage of the proposed model is promising. People in the traffic simulation area can get a valid and accurate car-following model. This model can also be used as a classifier for recognizing driving behaviors of surrounding drivers for human or autonomous drivers by determining their current state and its semantic cluster. In addition, due to its insightful nature, an intelligent car-following controller's design can also benefit from this model. Experiments demonstrate that such a controller can mimic a human's car-following behavior.

This chapter is organized as follows. Section 3.2 introduces car-following model identification. Section 3.3 discusses timed automata learning. Section 3.4 explains the methodology about state sequence clustering. In Section 3.5, experiments and comparisons with baselines are conducted. Another potential application of the proposed model is discussed in Section 3.6. The concluding remarks are made in Section 3.7.

3.2. CAR-FOLLOWING MODEL IDENTIFICATION

In this chapter, two commonly used models are introduced: the Helly and the IDM, which are representations of a linear car-following model and a non-linear one.

The acceleration in Helly's car-following model is a linear function combining the relative speed and the relative distance between the headway and the desired headway, which is defined by (Helly, 1959a):

$$\dot{v}(t) = C_1 \cdot \Delta v(t-\tau) + C_2 \cdot (\Delta x(t-\tau) - D(t)) \tag{3.1}$$

and

$$D(t) = \alpha + \beta \cdot v(t-\tau) + \gamma \cdot \dot{v}(t-\tau) \tag{3.2}$$

where C_1, C_2, α, β, γ and τ are the constant parameters that need to be calibrated. The desired headway is a function of the speed and the acceleration of the following vehicle, where α, β and γ are the corresponding weightings for those variables, and τ represents the reaction time of the following vehicle.

The acceleration in the IDM is a continuous function associated with the speed v, relative distance Δx, and relative speed Δv, which is defined by (Treiber et al., 2000):

$$\dot{v} = a_0 \cdot \left(1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*(v, \Delta v)}{\Delta x}\right)^2\right) \tag{3.3}$$

and

$$s^*(v, \Delta v) = s_0 + v \cdot T_0 + \frac{v \cdot \Delta v}{2\sqrt{a_0\, b_0}} \tag{3.4}$$

where a_0, b_0, v_0, δ, s_0 and T_0 are constant parameters that need to be calibrated. The exponential constant δ is usually set to 4. In Equation 3.3, the acceleration function is divided into two parts. The first part $a_0 \cdot \left(1-(v/v_0)^{\delta}\right)$ represents an acceleration rate toward a desired speed v_0, while a_0 denotes the maximum acceleration. The second part $-a_0 \cdot (s^*(v,\Delta v)/\Delta x)^2$ indicates a braking action according to the relative distance Δx and a desired minimum gap s^*, which is defined by Equation 3.4. b_0 and s_0 are the desired deceleration and the minimum safe distance, respectively. T_0 indicates the desired safety time gap.

In this chapter, the differential evolution algorithm (DEA) (Storn and Price, 1997) is applied to identify the parameters of the Helly and the IDM car-following models.

3

3.3. STATE MACHINE LEARNING

State machine learning, also known as grammatical inference (GI), aims at identifying a "correct" grammar for an unknown target language, given a finite number of examples of the language (Sakakibara, 1997). The main goal of grammatical inference is learning regular grammars or deterministic finite automata (DFA), typically minimum-state DFA (de La Higuera, 2005). The first convincing model for grammatical inference dates back to 1967 (Gold, 1967). It has been proven that finding the minimum-state DFA from incomplete examples is NP-complete (Gold, 1978a). Readers are referred to the survey paper (Stevenson and Cordy, 2014) for more formal definitions and a history of grammatical inference. Although GI is hard in theory, new techniques, e.g., heuristic-based state merging, have emerged to make practical problems more tractable (de La Higuera, 2005). These algorithms require discrete-event strings as input. In this chapter, the original real-valued time series data are abstracted using a symbolic representation associated with time information. The resulting timed strings are then fed to a state machine inference algorithm that learns a structural model, uncovering the underlying behaviors.

3.3.1. PROBABILISTIC DETERMINISTIC REAL TIMED AUTOMATON

A probabilistic deterministic finite automaton (PDFA), defined in Definition 1, is a generic model for discrete events (similar to a Hidden Markov Model).

Definition 1. A PDFA is a 5-tuple $\langle Q, \Sigma, \delta, \pi, q_0\rangle$, where Q is a finite set of states, Σ is a finite alphabet of observable symbols (events), $\delta : Q \times \Sigma \rightarrow Q$ is the transition function from a state-symbol pair to the next state, $\pi : Q \times \Sigma \rightarrow [0,1]$ is the probability of the emitted symbol given a state, and q_0 is the initial state.

Sequences of symbols translate to paths over states starting from the initial state q_0. The probability of such a sequence is obtained by multiplying all the state-symbol probabilities along such a path. Time information is also relevant in many real-world applications of automata. The actions' timing or lifetime is important for characterizing behaviors. Sharp and slow deceleration actions are conspicuously distinct for instance. An algorithm for efficient learning of timed automata was proposed in Ref. (Verwer et al., 2006, 2010a). This algorithm uses an *explicit* representation of such time constraints. Discrete events are represented by timed strings $(a_1, t_1)(a_2, t_2)\cdots(a_n, t_n)$, where a_i is a discrete event occurring with t_i time delay since the $(i-1)$th event. A probabilistic deterministic real timed automaton (PDRTA) model defines a probability distribution over such timed strings, having a Markov property in the distribution over events, and a semi-Markov property in the time guards. A PDRTA is formally defined in Definition 2.

Definition 2. A PDRTA is a 4-tuple $\langle \mathscr{A}, \mathscr{E}, \mathscr{T}, \mathscr{H} \rangle$, where $\mathscr{A} = \langle Q, \Sigma, \Delta, q_0 \rangle$ is a 4-tuple defining the machine structure: Q is a finite set of states, Σ is a finite alphabet, Δ is a finite set of transitions, and $q_0 \in Q$ is the initial state. $\mathscr{E}$ and $\mathscr{T}$ are the event and time probability distributions, respectively. $\mathscr{E} : Q \times \Sigma \to [0,1]$ returns the probability of generating/observing a given event in a given state. $\mathscr{T} : Q \times \mathscr{H} \to [0,1]$ returns the same but for a given time range $[m, m'] \in \mathscr{H}$, where $\mathscr{H}$ is a finite set of non-overlapping intervals in $\mathbb{R}_+$. A transition $\delta \in \Delta$ in a PDRTA is a tuple $\langle q, q', a, [m, m'] \rangle$, where $q, q' \in Q$ are the source and target states, $a \in \Sigma$ is a symbol and $[m, m']$ is a temporal guard.

In a PDFA and a PDRTA, the states are *latent variables* that cannot be directly observed in strings, but have to be estimated by using a learning method. The state transition in a PDFA is triggered only by an event. However, in a PDRTA, it is triggered when both an event and its timing are validated (inside a time range/guard). Therefore, a PDRTA is essentially a timed variant of a PDFA.

Example 2. Figure 3.2 illustrates an automaton modeling a simple driving scenario. Let us imagine that in the initial state $S0$, the subject vehicle keeps a large relative distance to the leading vehicle, which is speeding up. If the subject vehicle slows down, the relative distance will be greater and it will end the car-following period, i.e., ending in the state $S1$ (assume that $S1$ is a stable final state). For the state sequence $S0-S2-S1$, the subject vehicle keeps constant speed for a long time, with time constraints of 30-60 seconds, and afterwards slows down. It needs to be clarified again that the time in a TA is the time elapsed since the last event. As a consequence, it ends in state $S1$. The state sequence $S2-S3-S4$ and a more complete loop $S2-S3-S4-S2$ show typical car-following behaviors. The subject vehicle in these cases keeps a small relative distance and a small relative speed to the leading vehicle. The transition from $S2$ to $S3$, i.e., speed up $[0,10]$, denotes that after within 10 seconds of keeping a constant speed, the subject driver quickly speeds up and catches the leading vehicle. The probabilities next to transition arcs are the joint distribution of symbols and time constraint. The probability of a state sequence is therefore easy to compute, say the probability of $S0-S2-S3-S4$ is $0.8 \times 0.9 \times 1.0 = 0.72$.

3.3.2. DATA DESCRIPTION

This chapter uses the public dataset on individual vehicle trajectories from the Next Generation SIMulation (NGSIM) (NGSIM, 2007), a program funded by the U.S. Federal Highway Administration. The trajectory data provide a great and valuable basis for validation and calibration of microscopic traffic models (Thiemann et al., 2008). The I80 and the US101 are two datasets from Highways I80 and the US101, respectively.

The I80 dataset consists of three 15-minute periods: 4:00 p.m. to 4:15 p.m., 5:00 p.m. to 5:15 p.m., and 5:15 p.m. to 5:30 p.m. These periods represent the buildup of congestion, or the transition between uncongested and congested conditions, and full congestion during the peak period (NGSIM, 2007). A total of 45 minutes of data are available in the US101 dataset, which are segmented into three 15 minute periods: 7:50 a.m. to 8:05 a.m., 8:05 a.m. to 8:20 a.m., and 8:20 a.m. to 8:35 a.m. (NGSIM, 2007). Both the I80

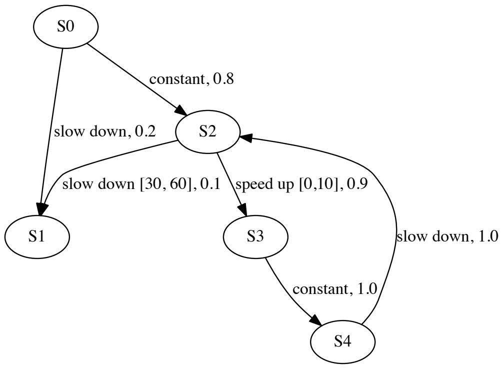


Figure 3.2: A simple example of the timed automaton computation. Note that it is only used for illustrating a timed automaton and some following techniques based on an already learned model. It is not a model learned from the dataset in the experiment using our algorithm.

and the US101 datasets provide precise trajectory information for each vehicle within the study area at a sampling frequency 10 Hz. The distribution of the time duration of car-following sequences in each dataset is illustrated in Figure 3.3.

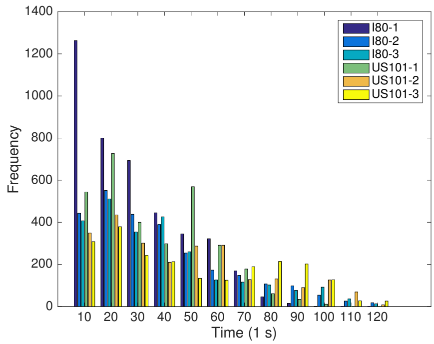


Figure 3.3: The duration distribution of car-following sequences in each dataset. The frequency on the y-axis is the number of sequences in each time bin.

Based on the trajectory data, the following and leading vehicle pairs are extracted for the purpose of studying car-following behavior. Note that vehicle speed, relative distance, and relative speed are explanatory variables as inputs. Longitudinal acceleration is a response variable as an output.

3.3.3. DATA PRE-PROCESSING

The k-means clustering algorithm is used as a discretization approach to symbolize the car-following data. The centroids of the I80-1 dataset are listed in Table 3.1. The

Table 3.1: Code book of the k-means centroids for numeric data in the I80-1 dataset.

Symbols	a	b	c	d	e	f	g	h	i	j
v centroid (m/s)	0.79	3.02	−2.88	4.82	−3.12	−0.98	−9.67	2.52	−7.02	0.12
Δx centroid (m)	57.87	36.13	15.63	15.55	204.18	96.09	39.74	24.00	24.47	10.13
v centroid (m/s)	13.69	10.54	7.74	5.94	19.41	17.25	12.99	8.38	10.10	4.12

"ELBOW" method is used to determine the "optimal" number of clusters (Goutte et al., 1999). The idea is to find the number of clusters that stops sharp dropping of the WSS (within the cluster sum of squares), which is illustrated in Figure 3.4. Symbolic strings are then converted to timed strings. Figure 3.5 shows a simplified example with the speed feature to illustrate how the conversion works. In the experimental setup, all 3 input features are clustered at once.

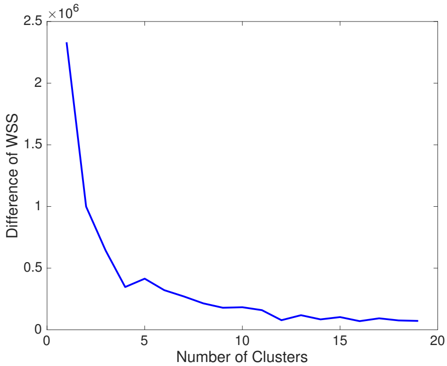


Figure 3.4: The WSS difference versus the number of clusters in I80-1. It is suggested that there is often a range of reasonable number of clusters to return, e.g., 9 to 12 in this case, rather than a single correct number (Salvador and Chan, 2005a). 10 is selected as a reasonable number of clusters.

3.3.4. LEARNING PDRTAS

A state-of-the-art machine learning algorithm named RTI+ is used to learn car-following behaviors from unlabeled data. For more details about this algorithm, readers are referred to the Ref. (Verwer, 2010a). Traditional state machine learning algorithms start by building a large tree-shaped automaton called an augmented prefix tree acceptor (APTA) from a sample of input strings. Every state of this tree can be reached by exactly one untimed string and therefore encodes exactly the input sample. For timed automaton learning, the initial values of the lower and upper bounds of all time guards are set to be the minimum t_{min} and maximum t_{max} time values from the input samples S. Figure 3.6 illustrates a timed APTA (TAPTA) from timed strings (a modified example from Ref. (Verwer, 2010a)).

State merges and transition splits are two main operations of structure and temporal guards learning in RTI+. A split of a transition (see an example shown in Figure 3.7) $\delta = \langle q, q', a, [m, m'] \rangle$ at time t creates two new transitions $\langle q, q_1, a, [m, t] \rangle$ and

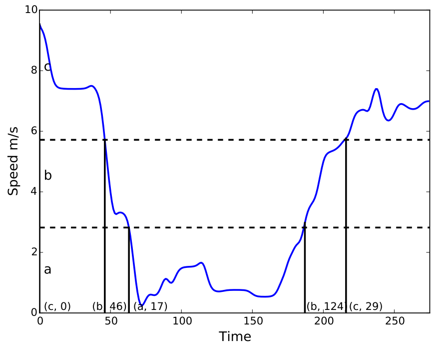


Figure 3.5: Discretization of time series data in I80-1. Instead of using complete symbolic strings with total length 275, the timed string has 5 tuples as input: $(c,0)(b,46)(a,17)(b,124)(c,29)$. The number next to the symbol in each tuple denotes the time difference since the last event.

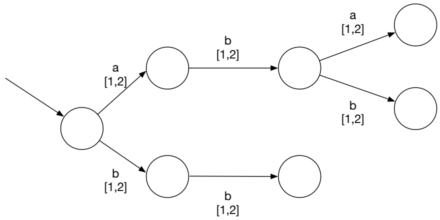


Figure 3.6: A TAPTA for the timed input sample: S=(a,1), (a,1)(b,2)(b,1), (b,2)(b,1), (a,1)(b,1)(a,1), (b,2),(b,1)(b,1)

$\langle q, q_2, a, [t+1, m']\rangle$. The target states q_1 and q_2 are the roots of two new prefix trees that are reconstructed based on the input sample.

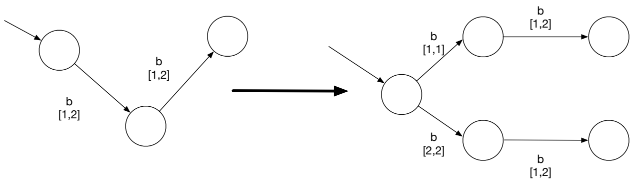


Figure 3.7: A split of a part of the TAPTA from Figure 3.6

The algorithm also greedily merges pairs of states (q, q') in this tree, forming a smaller and smaller machine that generalizes over samples, as shown in Figure 3.8. Because PDRTAs are deterministic, for every event $e \in \Sigma$ the states that are reached from q and q' have to be merged as well (the determinization process).

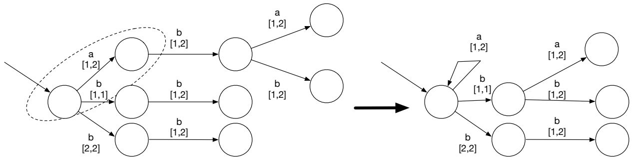


Figure 3.8: A merge operation of TAPTA after the split from Figure 3.7

Note that these examples are only one possible split and merge, illustrating how to conduct these operations. The algorithm uses a likelihood-ratio statistical test to decide whether to split/merge or not (Verwer et al., 2010a). A hypothesis H is called nested within another hypothesis H' if the possible distributions under H form a strict subset of the possible distributions under H'. By definition, H' has more unconstrained parameters (or degrees of freedom) than H ($r' > r$). In our case, H is the model after merge (resp. before a split) and H' is the model before a merge (resp. after a split). Given two hypotheses H and H' such that H is nested in H', and a data set S, the likelihood ratio test statistic is computed by:

$$LR = \frac{LK(S, H)}{LK(S, H')} \tag{3.5}$$

where the likelihood LK estimates how likely S is to be generated by the corresponding hypothesis. The random variable $y = -2\ln(LR)$ is asymptotically $\chi^2(r' - r)$ distributed (Wilks, 1938). The p-value is computed. If it is smaller than 0.05, H and H' are significantly different with 95% confidence so that a split operation is accepted. In addition, a merge is accepted whenever the model after the merge is not significantly different from the model before the merge since they are supposed to have similar or compatible stochastic and timed behaviors. Note that the current version of RTI+ tries to model time and events distributions independently. An overview of RTI+ is in Algorithm 4.

Algorithm 4 Data identification with RTI+:

Require: A (multi-)set of timed strings S_+
Ensure: A small PDRTA $\mathscr{A}$ for S_+
  Construct a timed prefix $\mathscr{A}$ tree from S_+, let $Q' = \emptyset$
  for all all transitions $\delta = \langle q, q', a, [m, m'] \rangle$ from $\mathscr{A}$, **do**
    Evaluate all possible merges of q' with states from Q'
    Evaluate all possible splits of δ
    if the lowest split p-value< 0.05 **then**
      perform this split
    else if the highest merge p-value> 0.05 **then**
      perform this merge
    else
      add q to Q'
    end if
  end for

3.4. STATE SEQUENCE CLUSTERING

Latent states are usually used for learning sequential patterns. They reduce the dimensionality of data. A large number of *observable variables* can be aggregated in a model to represent an underlying concept/behavior, making it easier to understand the data. With the help of the learned timed automaton, a mapping is built between the observable variables (time series data/symbolic data) and the *latent variables* (state sequences).

Table 3.2: Mapping between timed strings and state sequences.

	Timed strings	State sequences
Frame 1	(slow down, 0)	S0, S1
Frame 2	(slow down, 0)	S0, S1
Frame 3	(constant speed, 0), (slow down, 50)	S0, S2, S1
Frame 4	(constant speed, 0), (speed up, 10), (constant speed, 5)	S0, S2, S3, S4
Frame 5	(constant speed, 0), (speed up, 6), (constant, 15), (slow down, 5), (speed up, 4), (constant speed, 15)	S0, S2, S3, S4, S2, S3, S4
...	...	...

Table 3.2 shows the frames mapping between the input timed strings and the output state sequences of the RTI+ for the case in Figure 3.2. The subsequence clustering is performed on each state sequence. The cluster IDs are used to look up the associated symbolic transition, and look up the origin domain corresponding to the symbol, and obtain the associated raw values. A sequence of cluster IDs is assigned to the symbolic string and the raw time series data. Because it is only needed to follow the mappings backwards, this is called a (reverse) indices mapping. The piece-wise fitting model pa-

rameters are obtained in each individual cluster of time series data, as shown in Figure 3.1. The advantages of state sequence clustering over direct symbolic clustering are as follows:

1. States are latent variables determining the distribution of symbols. However, the mapping from symbols to states is not unique. As a result, behaviors are more identifiable with a state sequence. For the example in Figure 3.2, a symbolic pattern “constant speed-slow down" can be interpreted ambiguously as a quitting car-following behavior (the state sequence $S0-S2-S1$) or an adapting speed car-following behavior (the state sequence $S3-S4-S2$). The identification by using states avoids this problem.

2. Symbolic clustering without time information is not able to distinguish behaviors with short or long duration. This information is encoded with time guards of states in a timed automaton.

The final fitting error of the car-following models using a direct symbolic clustering is compared with the novel state clustering in the experiments.

3.4.1. COMMON STRINGS

The state frames dataset DS contains N state sequences, i.e., $DS = \{S_1, \cdots, S_N\}$, where $S_i = (s^i_1, \cdots, s^i_{L_i})$ is a single sequence of length L_i containing states from Q. A substring, also called a factor of a string S^i, is a string $\hat{S^i} = (s^i_{1+j} \cdots s^i_{m+j})$, where $j \geq 0$ and $m + j \leq L_i$. Given a DS, a frequent common substring problem is to find strings (not necessary the longest in this chapter) that occur as substrings of at least ϵ state sequences, where $2 \leq \epsilon \leq N$ is a user-defined threshold (Hirschberg, 1977). Intuitively, it is aimed at finding patterns that are shared among drivers as common frequent behaviors, which potentially characterize car-following behaviors.

3.4.2. HIERARCHICAL STRING CLUSTERING

The Jaro-score is used to measure the similarity between two strings (Cohen et al., 2003).

$$JS = \begin{cases} 0 & \text{if } N_{match} = 0 \\ \frac{1}{3}\left(\frac{N_{match}}{L_i} + \frac{N_{match}}{L_j} + \frac{N_{match}-N_T}{N_{match}}\right) & \text{otherwise} \end{cases} \tag{3.6}$$

where L_i and L_j are the respective lengths of these two strings. N_{match} is the number of matching characters that are not farther than a window length $\lfloor \frac{max(L_i,L_j)}{2} \rfloor - 1$. N_T is half the transpositions number. The higher the Jaro score is, i.e., closer to 1, the more similar two strings are. We use $d = 1 - JS$ as the metric measuring string distance. For the two state sequences *1,6,2* and *1,6,2,1*, for instance, $d = 1 - \frac{1}{3}\left(\frac{3}{3} + \frac{3}{4} + \frac{3-0}{3}\right)$. The 4th letter “1" in the second sequence does not match the 1st letter “1" in the first sequence, since its index distance is larger than the length of the matching window, i.e., 1 in this case.

A hierarchical clustering is deployed for frequent common strings using the Jaro-score as distance (Ushioda, 1996). At the beginning, every string represents a unique cluster, then a hierarchical clustering essentially conducts pairwise distance computation between two clusters. For clusters containing multiple strings, we compute the average distance. The complete iteration illustrated in Figure 3.9 is a dendrogram. In each iteration, only one pair of clusters is merged. The *cut-off* threshold, the black dashed line in Figure 3.9, is a user-defined parameter for determining the number of clusters. Similar to determining the alphabet size, an ELBOW analysis can be also applied to select a good threshold.

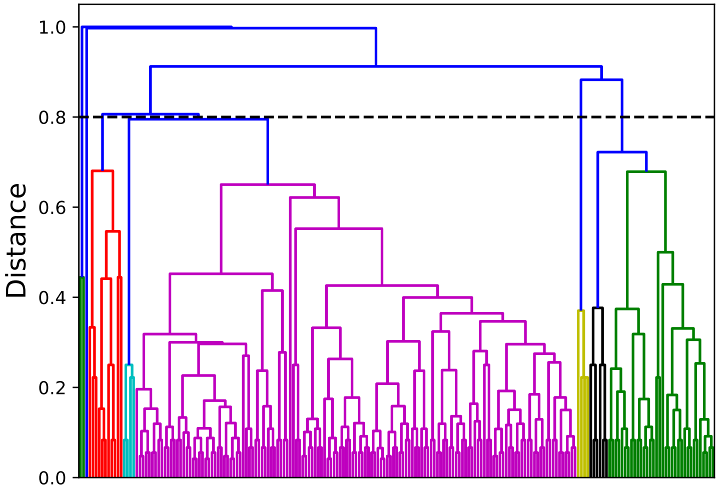


Figure 3.9: Hierarchical clustering of frequent sub-strings

The sequences in Table 3.2 are used to briefly explain how the subsequence clustering works step by step.

1. Extracting frequent common substrings:
   $S0, S1$;
   $S0, S2$;
   $S2, S3$;
   $S3, S4$;
   $S2, S3, S4$;
   $S0, S2, S3$.
   The support parameter of common strings ϵ is set to 2 in this case, and thus the substring $S2, S1$ will not be extracted as a frequent common substring because it only occurs in one state sequence.

2. Clustering substrings: for instance, we have 2 clusters after a hierarchical clustering:

Substring cluster 1: $S0, S1$;
Substring cluster 2: $S0, S2$; $S0, S2, S3$; $S2, S3$; $S3, S4$; $S2, S3, S4$.

3. Clustering states: States cluster 1: $S1$;
States cluster 2: $S2$, $S3$, $S4$;

3

$S0$ does not have to be classified as an initial state. Note that due to a different threshold setting or different ways of computing substring similarity, some states will be in multiple substring clusters (e.g., how to assign $S2$'s state cluster ID if $S0, S2$ is in substring cluster 1 instead of substring cluster 2). To avoid the ambiguity of states interpretation, ambiguous states are classified by an additional majority voting. For the aforementioned case of $S0, S2$ in substring cluster 1, $S2$ will be classified into state cluster 2 because the majority of $S2$ exists in the substring cluster 2. A new example arriving string $S0, S2, S3, S4, S2, S1$ is assigned state cluster IDs $2, 2, 2, 2, 1$ based on the aforementioned clusters obtained (again, the initial state is skipped).

3.4.3. ON-LINE INFERENCE

The states estimation is achieved online over arriving stream data. Starting with the initial state, observed numeric data will be first converted to symbols according to the numeric k-means codebook, say, the observation "constant" by the closest centroid computation. The state is transited from $S0$ to $S2$. The following transition is triggered until a new observation, like "speed up" or "speed down", occurs. The time difference is also computed between two consecutively distinct events. The state cluster ID and its corresponding car-following model is obtained as well because the state clusters have already been obtained in the states clustering step. Then the output (i.e., acceleration) of such a car-following model is computed from the input data (i.e., speed, relative speed, and relative distance). The generation of car-following traces includes one-step and multi-step approaches (Nippold and Wagner, 2012). The one-step approach evaluates the difference between the commutated output with the ground truth at each time point. The real status of the subject vehicle is updated from the dataset in the next time point, thus the error will not be accumulated in such a setting. The results of one-step testing are analyzed in Section 3.5. The multi-step generation only sets the initial state of the subject vehicle. During the generation procedure, real values of the subject vehicle in the dataset are not used to update its real-time information. The movement of the subject vehicle is updated completely using the computation model. The details of multi-step testing are discussed in Section 3.6. Note that in both settings, the trajectories of the leading vehicles are directly from the dataset.

3.5. EXPERIMENTAL RESULTS

The training and testing dataset split is listed in Table 3.3. In the following experiments, the k-means discretization and the state sequence clustering are both deployed only in the training dataset. To avoid over-fitting and obtain a less biased evaluation, the testing data are not included during clustering. Their symbolic and sequential labels are assigned by computing the closest distance to the clusters obtained from the training dataset. To make a complete overview of driving behaviors, the whole dataset is used for model interpretation. As a consequence, some settings in the training dataset, e.g., the thresholds and the number of clusters, are not necessarily the same as those in the whole dataset.

Table 3.3: Training and testing dataset

Dataset name	Proportion	Usage
Training set	80%	Symbols and state sequences clustering
Testing set	20%	Testing fitting error
Whole set	100%	Model interpretation

3.5.1. MODEL INTERPRETATION

One of the main advantages of the proposed method is that both the model and the clusters are interpretable. In this subsection, it will be shown how they can explain car-following behavior. The learned model from the whole I80-1 dataset is illustrated in Figure 3.10. All clusters are distinguished with different colors. There are loops with signifi-

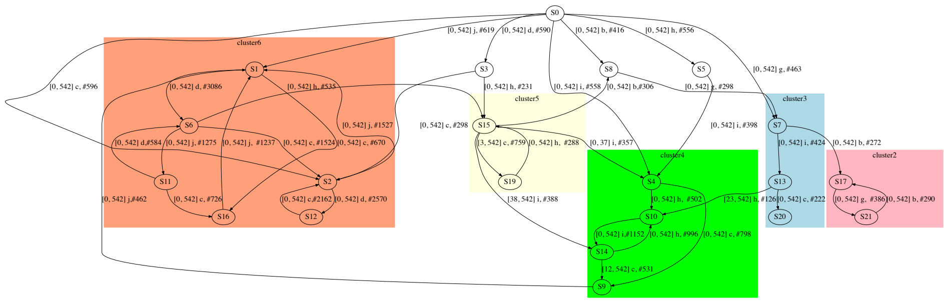


Figure 3.10: Real-timed automaton learned from the whole I80-1 dataset. Note that the original solution gotten from RTI+ has 34 states in total. The states with low frequencies are removed to simplify the model interpretation. For instance, states with event "e" occurring rarely are not shown in this figure. The arcs represent transitions between states. The information of timed guards, events, and number of occurrences is also printed next to the arcs.

Table 3.4: Interpretation of Clusters in the I80-1 Dataset

Cluster ID	Dominating states	Dominating symbolic loops	Description
1	Remaining states	-	Intermediate process and infrequent states
2	17, 21	b-g	Steady long distance car-following
3	7,13,20	-	Intermediate process
4	4, 9, 10, 14	h-i	Steady medium distance car-following
5	15, 19	-	Intermediate process
6	1, 2, 6, 11, 12, 16	c-d-j	Steady short distance car-following

cantly large occurrences in cluster 6, e.g., state sequence: $1-6-11-16-1$ with symbolic transitions loop: d-j-c-j. The relative distances of "c" and "d" are close (centroid values: 15.63 and 15.55, cf. Table 3.1), but having negative and positive relative speed, respectively. They are associated with "j", which has a small speed difference. This sequence can be interpreted as **steady car-following behavior at short distances**, i.e., adapting the speed difference with the lead vehicle around 0. An example is shown in Figure 3.11. Similarly interesting and significant loops can also be seen in Clusters 2 and 4, which are

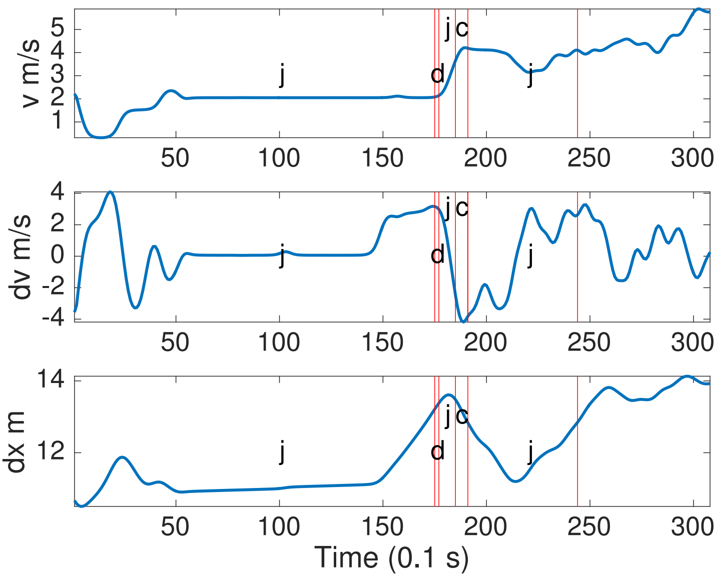


Figure 3.11: An example from one car-following sequence. The full timed string is j 0, d 175, j 2, c 8, j 6, d 53, j 12, d 34, j 5, c 2. Only the first 5 symbols are shown with j-d-j-c-j for simplification. You will see in the subplot of dv, the relative speed changes from positive (d) to small vale (j), negative (c), then j.

steady long distance and **steady medium distance car-following** behaviors respectively. An intermediate state $S15$ in Cluster 5 indicates how to transfer from Cluster 6 to other ones. For example, $S6-S15-S4$ with transitions "h, i", i.e., slowing down and speeding up to catch up, from the short distance following in Cluster 6 to the medium distance following in Cluster 4. The time split can also be seen in two branches of $[0,37], i$ and $[38,542], i$ from $S15$. They share the same symbolic transition condition but have distinct time guards. This means the "i" speed-up action followed by short or long duration of "h", i.e., after how much time the subject vehicle driver notices that his or her relative distance has been expanded by the leading vehicle and begins to catch up.

A complete car-following example in the I80-1 dataset is illustrated in Figure 3.12. It starts from the bottom (colored orange), passes through Clusters 6, 5, and 3, then finishes in Cluster 4. In the beginning, the subject vehicle is following the leading vehicle at short distances. Then the leading vehicle speeds up, see the positive relative speed and the increasing relative distance in Cluster 5. The subject vehicle then also speeds up to approach the leading vehicle, see the negative relative speed and the decreasing relative distance in Cluster 3. Finally, it follows the leading vehicle at medium distances in Cluster 4.[1] It can be seen that in Clusters 6 and 4, the subject vehicle enters an unconscious reaction region, also called a steady car-following episode, i.e., the relative distance and the relative speed are both bounded in a small area. Clusters 3 and 5 can both be treated as intermediate transition processes. Tracking the observed traces of a vehicle in the proposed model helps to understand its current status by looking at its state and semantic cluster.

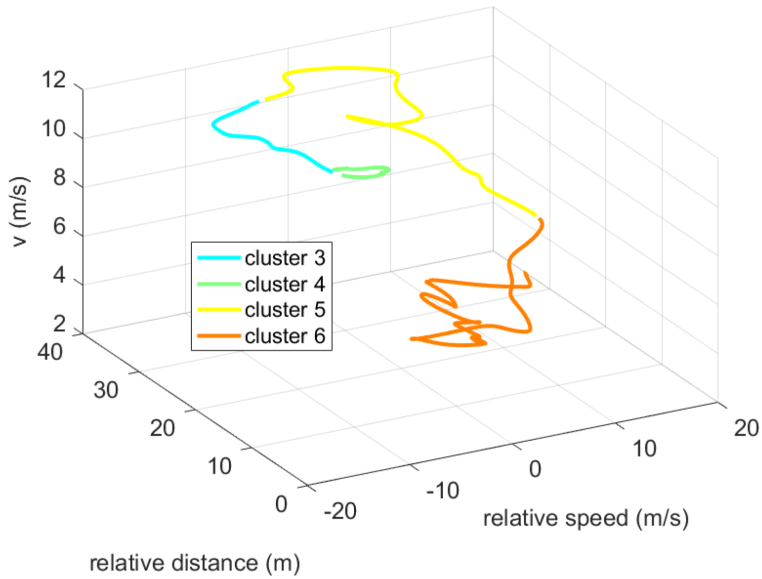


Figure 3.12: An example of complete car-following period switching among clusters in the I80-1 dataset.

3.5.2. COMPETING METHODS

Some baselines are implemented for comparisons with the proposed model. It will be explained how to implement them and why it is necessary to compare with them.

1. The first one is the ***gross fitting*** that uses a single car-following model. By comparing with it, it can be investigated how much improvement we can get using the clustering and the fitting with multiple models.

2. The second one is the ***symbolic clustering***. This comparison shows the value of clustering state sequences (latent variables) instead of clustering symbolic ones (observable variables) in identifying behaviors. The main idea is that the clustering is deployed with the same setting as the state sequence clustering, directly on the symbolic data without the time information. Note that the symbolic strings are

[1] An animated video can be found in our code repository: https://bitbucket.org/anjutalq/carfollowingrti/video

essentially timed strings without time information. This approach is applied because the original symbolic data sampled every 0.1s have too much redundancy, leading to large errors.

3. The third one is a competing state-of-the-art method proposed by ***Higgs*** et al. (Higgs and Abbas, 2015). The first step of their method is segmenting multi-variate time series data by minimizing their variance. The objective function in a normal Z-scale is defined as:

$$\min Z = \sum_{i=1}^{l}\sum_{j=1}^{n}\sum_{k=1}^{o}\frac{x_{ijk}-\bar{x}_{ij}}{S_{ij}} \tag{3.7}$$

subject to $t_i = [0, T]\,\forall i$ and $\sum t_i = T$, where

x_{ijk}, the kth observation of variable j in segment i;

$\bar{x}_{ij}$, centroid of segment i for variable j;

$\bar{S}_{ij}$, standard deviation of segment i for variable j;

l, number of segments in a car-following period;

n, number of variables;

o, number of observations in segment i;

t_i, length in time of segment i;

T, total length in time of a car-following period.

A bottom-up strategy is deployed to minimize the objective function. Initial segments are set with equal length (the same setting of 3 seconds is used from Ref. (Higgs and Abbas, 2015)). In each iteration, a pair of adjacent segments with the lowest merge cost (the largest reduction of Z value) is merged. The iterative process is terminated when the criterion is met, e.g., setting 10 as the maximum number of segments (Higgs and Abbas, 2015). Then, mean values representing segmented piece-wise data are clustered using k-means. To make a fair comparison, the number of clusters is set to be the same as the proposed approach.

4. The last baseline is called the ***state model***, which is also based on the learned timed automaton. Without partitioning the model into state clusters, individual models are trained in each state. This inevitably introduces a large number of models and their parameters but helps us to investigate the benefit of clustering states.

The root-mean-square error (RMSE) is a widely used indicator for evaluating the acceleration error of car-following models. In addition, to overcome overestimation in high and low values, some papers (Kesting and Treiber, 2008; Chen et al., 2010) use speed's *relative error* (RE) $F_{rel}(\text{vel})$, *absolute error* (AE) $F_{abs}(\text{vel})$, and *mix error* (ME) $F_{mix}(\text{vel})$ as additional indicators, which are defined as follows:

$$F_{rel} = \sqrt{\left\langle \left(\frac{s^{sim}-s^{real}}{s^{real}}\right)^2 \right\rangle} \tag{3.8}$$

$$F_{abs} = \sqrt{\frac{\left\langle \left(s^{sim} - s^{real}\right)^2 \right\rangle}{\left\langle s^{real} \right\rangle^2}} \tag{3.9}$$

$$F_{mix} = \sqrt{\frac{1}{\left\langle \left|s^{real}\right| \right\rangle} \left\langle \frac{\left(s^{sim} - s^{real}\right)^2}{\left|s^{real}\right|} \right\rangle} \tag{3.10}$$

where s^{sim} and s^{real} are the computed and the real values respectively. $\langle s \rangle$ is the average value defined as $\langle s \rangle = \frac{1}{N} \sum_{i=1}^{N} s(i)$.

Table 3.5 and Table 3.6 show the comparison using the aforementioned indicators and their standard deviations. The best model is highlighted with the smallest mean error using a bold font. The variance is compared additionally when two models have the same mean error values. The average improvement of the proposed method over the gross fitting is summarized in Table 3.7 and Table 3.8. Note that here a single-step approach is deployed for both training and testing. A multi-step approach will be also tested for a trajectory simulation in Section 3.6. The single-step approach focuses on the deviation at each step in the time series data and represents the local calibration, while the multi-step approach focuses on the deviation of whole traces and represents a trajectory calibration. The difference lies in the fact that the input/output of the training system can be data point pairs (single values) or vector pairs (trajectories). Readers are referred to Ref. (Nippold and Wagner, 2012) for a more detailed explanation. The runtime of different approaches is also compared in Table 3.9. The training using the DEA costs most time on the Intel 3.1GHz i7 processors. The remarks and analyses are as follows.

- The proposed model and the state model are the best two in the results. In every dataset, they outperform gross fitting and the other two clustering models. Both of them are based on the learned timed automata. The only difference lies in the presence or absence of state clustering. The state model uses much more car-following models, e.g., 34 models in a 34-state automaton. The training of a state model, however, does not take too long since the data are split over many more models, and fewer data lead to a fast convergence. The overfitting problem of a car-following model calibration has been reported recently (Van Hinsbergen et al., 2015). Such overfitting is due to too few data during the training phase. To balance the bias (fitting error) and the variance (model complexity), it is suggested to use the proposed model with high accuracy, low complexity, and sufficient data per model.

- The symbolic method is the third best model, with competing performance to the proposed model. However, such a model-free pattern mining method can only serve as a clustering tool rather than a control model generating the following vehicle's trajectories.

- The RMSE of acceleration is a more sensitive indicator of the larger magnitude. Due to an integral relation from acceleration to speed, the speed's error has been smoothed and thus has a smaller magnitude. In addition, the testing is essentially a one-step prediction evaluation, i.e., the error will not be accumulated. Therefore, the improvement is less obvious than the multiple-step prediction.

- The symbolic labeling, the timed automata learning, and the sequence clustering are quite efficient in computation cost. They are promising in car-following model calibration on large-scale data.
- Among all the clustering methods, the Higgs model takes the longest time on clustering since the segmentation is time-consuming.

Table 3.5: Testing data error in NGSIM datasets: Helly Model

Mean ± Std.		Helly					
		I80-1	I80-2	I80-3	US101-1	US101-2	US101-3
RMSE (acc) m/s^2	Gross	0.9981±0.3343	1.4641±0.3971	1.6424±0.3754	1.6429±0.2859	1.5413±0.3051	1.3454±0.3402
	Symbolic	0.9319±0.3218	1.3774±0.3623	1.5648±0.3836	1.6005±0.2916	1.3656±0.2170	1.3012±0.2812
	Higgs	1.0999±0.5240	1.4207±0.3743	1.6216±0.3693	1.6273±0.2999	1.4753±0.3402	1.3220±0.2971
	Proposed	0.9225±0.3156	1.3659±0.3653	1.5552±0.3714	1.5962±0.2875	1.3452±0.2054	1.2984±0.2767
	State Model	**0.9122±0.3231**	**1.3648±0.3629**	**1.5541±0.3778**	**1.5899±0.2781**	**1.3405±0.2064**	**1.2962±0.2775**
F_{rel}(vel) m/s	Gross	0.0943±0.2848	0.0278±0.0103	0.0339±0.0194	0.0450±0.0949	0.0315±0.0682	0.0451±0.0780
	Symbolic	0.0145±0.0080	0.0267±0.0103	**0.0269±0.0100**	0.0186±0.0081	0.0178±0.0068	0.0245±0.0107
	Higgs	0.0507±0.0304	0.0266±0.0506	0.0274±0.0555	0.0355±0.0689	0.0250±0.0399	0.0443±0.0511
	Proposed	0.0146±0.0082	**0.0265±0.0102**	**0.0269±0.0100**	0.0188±0.0085	0.0178±0.0068	**0.0244±0.0106**
	State Model	**0.0144±0.0081**	0.0266±0.0104	0.0269±0.0101	**0.0185±0.0081**	**0.0177±0.0068**	0.0244±0.0107
F_{abs}(vel) m/s	Gross	0.0148±0.0095	0.0257±0.0092	0.0412±0.0244	0.0278±0.0097	0.0199±0.0049	0.0329±0.0115
	Symbolic	0.0127±0.0059	0.0245±0.0092	0.0256±0.0097	0.0165±0.0060	0.0164±0.0048	0.0207±0.0079
	Higgs	0.0159±0.0092	0.0346±0.0179	0.0360±0.0258	0.0206±0.0106	0.0180±0.0055	0.0290±0.0143
	Proposed	0.0128±0.0059	**0.0243±0.0091**	**0.0256±0.0095**	0.0166±0.0061	**0.0156±0.0047**	0.0201±0.0075
	State Model	**0.0126±0.0060**	0.0243±0.0092	0.0256±0.0096	**0.0164±0.0059**	0.0177±0.0068	**0.0200±0.0075**
F_{mix}(vel) m/s	Gross	0.0184±0.0194	0.0261±0.0092	0.0422±0.0168	0.0291±0.0109	0.0219±0.0064	0.0348±0.0128
	Symbolic	0.0132±0.0063	0.0250±0.0092	0.0258±0.0093	0.0170±0.0062	0.0172±0.0052	0.0219±0.0081
	Higgs	0.0166±0.0099	0.0367±0.0211	0.0372±0.0221	0.0216±0.0114	0.0188±0.0063	0.0314±0.0161
	Proposed	0.0133±0.0063	**0.0248±0.0091**	0.0258±0.0092	0.0171±0.0064	0.0162±0.0051	**0.0211±0.0078**
	State Model	**0.0131±0.0064**	0.0249±0.0092	**0.0257±0.0093**	**0.0169±0.0061**	**0.0161±0.0051**	**0.0211±0.0078**

Table 3.6: Testing data error in NGSIM datasets: IDM Model

Mean ± Std.		IDM					
		I80-1	I80-2	I80-3	US101-1	US101-2	US101-3
RMSE (acc) m/s^2	Gross	1.0917±0.8706	1.4327±0.3938	1.6060±0.4151	1.6334±0.4064	1.4801±0.2717	1.3180±0.2793
	Symbolic	0.9857±0.4282	1.3610±0.4298	1.5341±0.3654	1.5563±0.2550	1.3875±0.1992	1.2964±0.2524
	Higgs	1.0679±0.7976	1.3871±0.3972	1.5860±0.4262	1.5862±0.2578	1.4594±0.2041	1.3025±0.3390
	Proposed	**0.9798±0.4340**	1.3280±0.3908	**1.5289±0.3659**	**1.5555±0.2567**	**1.3634±0.1992**	**1.2944±0.2497**
	State Model	1.0174±0.4718	**1.3254±0.3810**	1.5332±0.3842	1.5583±0.2510	1.3966±0.2693	1.2971±0.2690
F_{rel}(vel) m/s	Gross	0.0799±0.2204	0.0360±0.0108	0.0338±0.0191	0.0419±0.0936	0.0514±0.0916	0.0572±0.1030
	Symbolic	0.0159±0.0083	0.0262±0.0107	0.0265±0.0102	0.0180±0.0080	0.0185±0.0071	0.0293±0.0104
	Higgs	0.0468±0.0335	0.0302±0.0734	0.0265±0.0507	0.0355±0.0689	0.0210±0.0117	0.0481±0.0713
	Proposed	**0.0152±0.0086**	**0.0261±0.0109**	0.0264±0.0102	**0.0180±0.0080**	**0.0179±0.0070**	**0.0239±0.0104**
	State Model	0.0153±0.0082	**0.0261±0.0109**	**0.0264±0.0101**	0.0181±0.0081	0.0182±0.0071	0.0241±0.0106
F_{abs}(vel) m/s	Gross	0.0263±0.0119	0.0297±0.0093	0.0410±0.0238	0.0174±0.0094	0.0171±0.0072	0.0285±0.0122
	Symbolic	0.0137±0.0065	0.0240±0.0095	0.0253±0.0098	0.0160±0.0057	0.0162±0.0049	0.0210±0.0077
	Higgs	0.0234±0.0137	0.0308±0.0184	0.0385±0.0295	0.0175±0.0089	0.0170±0.0059	0.0267±0.0135
	Proposed	**0.0135±0.0067**	**0.0237±0.0094**	**0.0251±0.0098**	**0.0159±0.0057**	**0.0157±0.0049**	**0.0200±0.0077**
	State Model	0.0139±0.0068	**0.0237±0.0094**	**0.0251±0.0098**	0.0160±0.0057	0.0160±0.0052	0.0201±0.0077
F_{mix}(vel) m/s	Gross	0.0289±0.0173	0.0343±0.0094	0.0431±0.0166	0.0187±0.0106	0.0192±0.0112	0.0346±0.0130
	Symbolic	0.0142±0.0067	0.0245±0.0094	0.0254±0.0093	0.0165±0.0060	0.0173±0.0053	0.0216±0.0079
	Higgs	0.0233±0.0131	0.0328±0.0201	0.0385±0.0210	0.0190±0.0102	0.0172±0.0061	0.0360±0.0150
	Proposed	**0.0139±0.0069**	**0.0243±0.0095**	**0.0252±0.0093**	**0.0165±0.0060**	**0.0163±0.0053**	**0.0210±0.0079**
	State Model	0.0141±0.0067	**0.0243±0.0095**	**0.0252±0.0093**	**0.0165±0.0060**	0.0166±0.0055	**0.0210±0.0079**

Table 3.7: Summary of improvement in each dataset: Helly model

Percentage of improvement (%)	I80-1	80-2	80-3	US101-1	US101-2	US101-3
RMSE (acc)	7.57	6.71	5.31	2.84	12.7	3.50
F_{rel} (vel)	84.52	4.68	20.65	58.22	43.49	45.90
F_{abs} (vel)	13.51	5.45	37.86	40.29	21.61	38.91
F_{mix} (vel)	27.72	9.96	20.85	41.24	26.03	39.37

Table 3.8: Summary of improvement in each dataset: IDM model

Percentage of improvement (%)	I80-1	80-2	80-3	US101-1	US101-2	US101-3
RMSE (acc)	10.25	7.31	4.80	4.77	7.88	1.79
F_{rel} (vel)	80.98	27.50	21.89	57.04	65.18	58.22
F_{abs} (vel)	48.67	20.20	38.78	8.62	8.19	29.82
F_{mix} (vel)	51.90	29.15	41.53	11.76	15.10	39.31

3.6. A HUMAN-LIKE CRUISE CONTROLLER

A valid car-following model is of great importance for traffic simulation. Besides that, the proposed model is promising in many other application scenarios. A human-like automatic cruise control system design will be discussed in this section. Other potential applications will be mentioned briefly in the future work.

The drawbacks of an automatic cruise control (ACC) system lie on an inconsistency between systems and human drivers (Hiraoka et al., 2005), because the control algorithm of an ACC focuses more on mathematical optimization of safety or comfort rather than driving behaviors. A valid car-following itself can be used as a controller which mimics real drivers' behaviors to avoid the inconsistency problem in a conventional ACC system.

The main idea of a human-like ACC system or a behavior simulator is learning a timed automaton from a real car-following training dataset and generating trajectories in a testing dataset. The position error of simulated traces and the real ones is evaluated in the testing dataset. The generation steps are as follows:

1. The subject vehicle starts from the initial state.

2. The speed, relative speed, and relative distance are computed on the fly. Note that we only control the following vehicle, i.e., the trajectory of the lead vehicle is directly from the dataset.

Table 3.9: Comparison of runtime

Models	Symbolic labeling (s)	Automata learning (s)	Clustering (s)	Training (s)	Testing (s)	Total (s)
Gross	-	-	-	488.24	3.84	492.08
Symbolic	69.72	-	53.75	2653.35	53.98	2830.80
Higgs	-	-	832.89	1534.62	33.95	2401.46
Proposed	69.72	16.09	24.56	2054.52	14.41	2179.30
State model	69.72	16.09	-	1690.41	22.36	1798.58

3. The current cluster of the subject vehicle is determined by its current state using the online inference discussed in Section 3.4.3, and then the parameter of the car-following model is selected to generate the desired acceleration.

4. The status of the subject vehicle, including speed, relative speed, and relative distance is continuously updated online using the acceleration computed in the last time step as well as the information of the lead vehicle from the dataset.

This approach is compared with a standard PID controller. The results of comparing position error in Table 3.10 show that the proposed model outperforms others.

Table 3.10: Comparison of Simulated Trajectory

Indicators	Proposed	Gross	PID controller
F_{rel} (vel) m/s	**0.1157±0.0807**	0.1332±0.0796	0.2466±0.2852
F_{abs} (vel) m/s	**0.0764±0.0643**	0.1091±0.0850	0.1105±0.0875
F_{mix} (vel) m/s	**0.0766±0.0615**	0.1034±0.0781	0.1360±0.0973

In this chapter, the simulated behaviors are learned from a large population of drivers' car-following data. However, it is possible to learn such a controller from a single driver if his/her data are sufficient. This is a promising approach for designing a specified car-following controller for an individual driver. Another advantage of our model is an active control strategy, e.g., forcing a state switching from short-distance following to a medium distance in the automaton.

3.7. CONCLUSION

In this chapter, a timed automaton model is learned from multivariate time series car-following data using a timed and symbolic representation. The model is easily visualizable and interpretable for the study of car-following behaviors. Sequential feature-based clustering of state sequences is used for partitioning the model to represent distinguishable behaviors. The original time series data are also clustered correspondingly. Different models are trained from individual clustered data to obtain a divide-and-conquer learning. Experiments demonstrate that the proposed method achieves high model fitting accuracy. Besides the general usage in traffic simulation, the proposed model can be used for subject drivers' decision-making by recognizing or predicting surrounding vehicles' car-following states and designing a more human-like car-following controller.

The imperfections of the proposed method include two aspects. First, compared with classic methods, the proposed model has higher complexity, though all processing steps can be automated. To some extent, it is not suitable for a fast traffic simulation with a relatively low precision. Second, from safety perspective, a data-driven design of an ACC system lacks theoretical guarantees, because it might be learned from poorly skilled drivers, though the proposed model is indeed an averaging model learned from thousands of human drivers. This problem will be overcome by a model checking technique (Henzinger et al., 1997b) or a supervisory control (Brandin and Wonham, 1994) with safety specifications.

In the near future, we will investigate more application cases by applying an automata learning lens. First, we can provide a visualizable model learned from traffic data of roads under observation. This helps insightful analysis of traffic flow situations. For instance, a congested traffic scenario should intuitively have many symbols indicating low speed in our model. Some intermediate process states in the proposed model somewhat reflect properties of traffic flow which deserve further investigation. Second, by observing the driving status of nearby vehicles, behaviors like steady car-following or approaching another vehicle can be recognized in our model. This is helpful for better perception of the subject vehicle.

4

LEARNING AUTO-REGRESSIVE DYNAMICAL MODELS USING REGRESSION AUTOMATA

In the last chapter we discussed the composed learning strategy for hybrid automata, where the discrete model learning and numeric model learning are actually separated. In this chapter, we will discuss another strategy called incline learning by using numeric data in addition to symbolic values for state machine learning. This novel type of syntactic model called regression automata and its learning algorithm are used for univariate time series modeling and forecasting.

The material in this chapter has appeared in
Qin Lin, Christian Hammerschmidt, Gaetano Pellegrino, and Sicco Verwer. Short-term time series forecasting with regression automata. In ACM SIGKDD 2016 Workshop on Mining and Learning from Time Series (MiLeTS), 2016

4.1. INTRODUCTION

Forecasting is one of the most significant challenges in time series analysis (Cryer and Kung-sik Chan, 2008; De Gooijer and Hyndman, 2006). In this chapter, we propose a novel model for learning syntactic patterns and forecasting time series. We apply our algorithm to wind energy prediction problems (Lin and Wang, 2014; Gu et al., 2015; Lin et al., 2013).

During the past 30 years, many methods for time series prediction have been proposed. Generally those techniques can be classified into two categories. The first one is the conventional statistical model. Autoregressive Moving Average (ARMA) is the most representative (Torres et al., 2005). Another conventional model is Kalman filter algorithm (Bossanyi, 1985). The techniques of second category are from the area of artificial intelligence and machine learning. The typical models that have been successfully applied in time series forecasting are neural networks (Guo et al., 2012), support vector machines (Mohandes et al., 2004), and fuzzy logic models (Damousis et al., 2004) to name a few.

Syntactic models are alternatives to the conventional systems, because the learned models allow one to inspect, interpret, and understand complex system dynamics (Albus et al., 2012; Hammerschmidt et al., 2016). Examples of such models are hidden Markov models (HMMs) and finite automata (FA) (MacDonald and Zucchini, 1997). Syntactic methods are based on symbols that have typically been abstracted from numeric data in a pre-processing step. This gives three main advantages: firstly, categorical prediction reduces the computation cost. Secondly, raw time series data in practice tend to be noisy. Symbolic representations are more robust to noise. Lastly, the category bounds can be modified to reflect prediction uncertainty, which is now becoming a trend in regression. To the best of our knowledge, the only syntactic models applied in wind speed prediction are Markov chains (Sahin and Sen, 2001) and semi-Markovian variants (D'Amico et al., 2014). An interesting indirect approach to syntactic modeling of daily foreign exchange rates was proposed by Lee Giles et al. (Giles et al., 2001). They first abstracted the raw financial data into symbols using a SOM (self-organizing map), and then applied RNNs (recurrent neural networks) to the sequences for training. Finally, DFA (deterministic finite state automata) were extracted from RNNs for model interpretation. Unfortunately, this novel model was only able to be used to predict directionality, i.e., whether the exchange rate is positive or negative in the future. Another related work is SAX (Symbolic Aggregate approXimation), which provided high-level representation for time series data (Lin et al., 2007). However the main goals of SAX were dimensionality reductions and similarity measurements rather than forecasting.

Syntactic models are useful because they provide a concise overview of numeric time series' behavior. A problem, however, is that they predict symbols instead of numeric values. Consequently, both their learning and prediction processes are less exact than those used by numeric models and therefore more difficult to evaluate and harder to use in practice. In this chapter, we overcome this problem of syntactic models by incorporating the numeric data values in the learning and prediction processes. Intuitively, the inputs of our model are the tuples of real numerical values and symbolical values abstracted from the raw data. The symbols are used for building the syntactic models underlying a time series' behavior at a high level with state transitions, while the numeric values are used to accurately reflect the evolution of time series.

We preprocess the raw time series data sequentially and discretize the numeric values into abstract symbols. We then learn an RA using the DFASAT algorithm (Heule and Verwer, 2013), but with a novel heuristic and a novel consistency criterion. Finally, we compare the resulting numeric predictions with baseline methods such as persistence, autoregressive integrated moving average (ARIMA), neural networks, and regression trees. The results demonstrate that our new method is competitive with these commonly used methods. Furthermore, they show that the numeric and syntactic prefix tree model used as input for DFASAT is already competitive with the state of the art, albeit worse than the model obtained after learning. This result demonstrates the power of our method used to combine numeric and syntactic data for time series prediction.

Our contributions are the following:

- We develop a new method for learning DFA from time series data using both numeric and symbolic inputs. To the best of our knowledge, this is the first work that proposes to learn automata for numeric regression tasks.
- We propose a novel heuristic and consistency test for guiding the automaton learning process.
- We show that the learned models make predictions in real unseen data with high accuracy, outperforming the competition in an application problem of short-term wind power prediction.

This chapter is organized as follows. Section 4.2 introduces data preprocessing, the model building, and the learning algorithm. The experimental results are presented in Section 4.3. Section 4.4 discusses the results and concludes the chapter.

4.2. DATA PREPROCESSING

4.2.1. DISCRETIZATION

The numeric signal needs to be abstracted as symbols for state machine (automaton) learning. In this chapter, we use SAX to discretize numeric data. Figure 4.1 illustrates an example of SAX. It firstly normalizes the raw data, then compresses by aggregating into piecewise aggregate approximations (PAAs). Lastly PAAs are assigned to symbols with quantiles of standard normal distribution. In this example, the raw data have length 48, the PAAs, i.e., colored bars have the same size of 12. We will finally get a frame with 4 letters “ccac”. If we SAX the whole training data set in the beginning and then slice them into frames, we will call this strategy as “global SAX” in this chapter. Table 4.1 shows the symbols and their corresponding numeric guards in the experimental case study one (see Section 4.3.3). All numeric values are abstracted to the symbol according to the bins they fall in. Note that we transform the bins of quantiles from standard normal distribution to un-normalized value for better explanation. We use an idea similar to that of “ELBOW" method (Goutte et al., 1999) to determine the “optimal” number of clusters, i.e., the alphabet size of SAX. The idea is finding the number of clusters that stops sharp dropping of the WSS (within cluster sum of squares), which is illustrated in Figure 4.2.

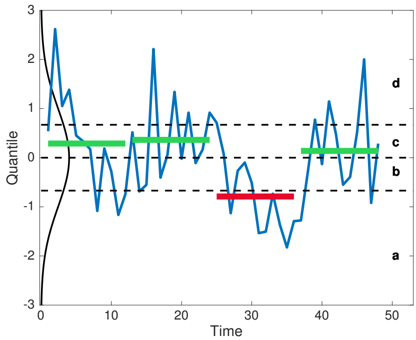


Figure 4.1: SAX labeling of time series data. The dashed lines indicate discretization boundaries.

Table 4.1: Global SAX guards for the wind speed prediction task, values are in m/s.

Symbol	a	b	c	d	e	f	g	h
Guard	(-∞, 0.59)	[0.59, 1.16)	[1.16, 1.58)	[1.58, 1.96)	[1.96, 2.34)	[2.34, 2.76)	[2.76, 3.33)	[3.33, +∞)

4.2.2. STATIONARITY AND DRIFT MODEL

Many time series in practice, such as the economic process and the wind speed, are difficult to predict since they are not stationary. Intuitively, the statistical properties of these processes, such as mean and variance, vary over time (Cryer and Kung-sik Chan, 2008). Logarithm and differencing are two widely used preprocessing methods for non-stationary time series (Cryer and Kung-sik Chan, 2008). The logarithm is useful to stabilize the variance of a time series of which larger values tend to have larger variance; meanwhile it helps to expand the difference of small values around zero. Differencing (1-st order derivative), i.e., computing the differences between consecutive observations, is useful to stabilize the mean of a time series by removing changes in the level of a time series, and so eliminating trend and seasonality. Assume that the original data of length N is $\mathbf{X} = [x_0, x_1, \cdots, x_{N-1}]$, and our goal is to get a drift model,

$$x_t - x_{t-1} = \hat{c} + e_t \tag{4.1}$$

where $\hat{c}$ is our estimated mean value of the drift, and e_t is assumed as white noise. Unlike the conventional time series models that directly take all the historic difference values into account to estimate $\hat{c}$, our syntactic model discovers patterns sharing similar behaviors to individually get the estimations of $\hat{c}$. Once $\hat{c}$ is learned from training data, Equation 4.1 is also used for forecasting with a known previous value.

Table 4.2: k-means centroids for the wind speed prediction task, values are in m/s.

Symbol	a	b	c	d	e	f	g	h
Centroid	0.76	1.22	1.68	2.20	2.82	3.63	4.81	7.46

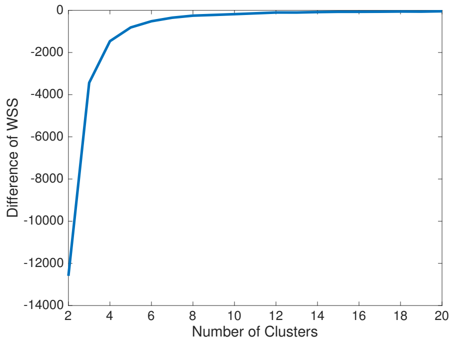


Figure 4.2: WSS difference versus number of clusters in training data. We select 8 as a good number of clusters.

Apart from global SAX and differencing, we also investigate the following strategies of preprocessing, and compare the results in the experiment (see Section 4.3.3).

- **local SAX** aggregates, discretizes, and normalizes data in each sliding window, see (Lin et al., 2007) for details.
- **k-means** with the same alphabet size as SAX is listed in Table 4.2, which shows the centroids of the symbols obtained in experimental case study one (see Section 4.3.3). All numeric values are abstracted to the symbol with the closest associated centroid.
- **logarithm differencing** compute the logarithm difference between consecutive observations, which actually reflects the ratio relations.

4.2.3. REGRESSION AUTOMATA

We provide a concise description of DFAs; the reader is referred to (Sudkamp, 2006) for a more elaborate overview. A *deterministic finite automaton (DFA)* is a quadruple $A = \langle Q, T, \Sigma, q_0 \rangle$ where Q is a finite set of states, $T : (Q, \Sigma) \to Q$ are labeled transitions with labels coming from an *alphabet* Σ, and $q_0 \in Q$ is the start state. A DFA computation starts in the start state q_0 and traverses transitions according to a given input string (sequence) $s_1 \dots s_n \in \Sigma^*$. At every index $1 \le i \le n$, the current state of the DFA is changed from source state q_{i-1} to target state $T(q_{i-1}, s_i)$. This computation is called deterministic because there exists exactly one target for every source-symbol pair. In contrast to the commonly used HMMs (Rabiner, 1989b), the computation path of a given DFA is thus completely determined for a given input string. This property makes them easier to learn. Learning DFAs is, however, much harder than learning Markov chains because (like HMMs) the traversed states are unknown (hidden) when given only input data.

A regression automaton (RA) is a quintuple $A = \langle Q, T, \Sigma, q_0, P \rangle$ where $\langle Q, T, \Sigma, q_0 \rangle$ is a DFA, and P is a prediction function $P : Q \to \mathbb{R}$. The prediction function assigns a prediction value to every state $q \in Q$. The computation of an RA is identical to that of a DFA,

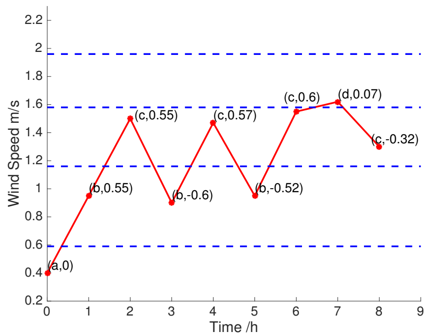


Figure 4.3: Our labeling of time series data, consisting of symbols and difference values. The dashed lines indicate discretization boundaries (using the code book in Table 4.1). To avoid redundancy of data, the values in this plot have been aggregated by SAX.

any numeric input data are ignored. Whenever a computation is in a state q, the value $P(q)$ is only used as a prediction for the next numeric data value. In our case, we use the preprocessing described above to obtain discretized symbols based on a time series signal, and numeric values based on the difference of the series, see Figure 4.3. The state of an RA is thus fully determined by the syntactic data, and the predicted drift value only depends on the current state. RAs can be seen as mappings from symbolic sequences to drift values.

4.2.4. EVIDENCE-DRIVEN STATE-MERGING

The current state of the art in DFA learning is evidence-driven state-merging in the red-blue framework (EDSM) (Lang et al., 1998), possibly with some search procedure (see, e.g., (Heule and Verwer, 2013)) in order to continue searching once a possible local optimum has been reached. In the following, we briefly explain the main steps of this algorithm together with our adaptations needed to handle numeric data.

PREFIX TREE CONSTRUCTION

The first step in EDSM is to build a Prefix Tree (PT) from the training data. For each input sample w from the training data, a chain is created by introducing a state between each letter w_i $(1 \leq i \leq |w|)$. This chain is inserted into the PT by traversing its labeled transitions until the word is fully inserted, or a leaf is reached. Upon reaching the leaf, the remaining sequence is appended at this position. For every state q in a PT, there exists exactly one computation that reaches q. A PT therefore encodes exactly the information in the (syntactic) training data, without any generalization. The set of states Q is extended to contain a null state $q_\perp$, to represent transitions for which no input data exist in the training sample, i.e., $T(q,l) = q_\perp$ means it is currently unknown what the target state is from state q with label l.

For RAs, the PT structure is constructed in the standard way using only the syntactic data, see Figure 4.4 for an example. The transitions are labeled with the symbol corresponding to the chosen discretization. In addition to the prefix tree structure, we aggregate the numeric values of all outgoing transitions in each node; the numeric values above states q_1, q_3, q_5 and q_8 are the average values of the differences of all outgoing transitions. If we want to predict the next value following 1.3, i.e., the original value of the last datum in Figure 4.3. we follow the transitions with the corresponding symbolic label, e.g., c, from the starting state q_0. In our example, it will transition to state q_5. By applying the reverse translation from Equation 4.1, the predicted value is $1.3 - 0.35 = 0.95$.

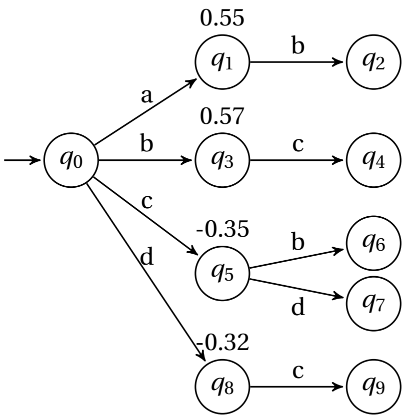


Figure 4.4: APTA for regression automata

MERGING STATES IN EDSM

The PT, encoding all the training data without generalization, usually leads to high-variance models sensitive to noise, and has an increased risk of overfitting. The goal of *DFA learning* is to find a *smallest* DFA A that is *consistent* with the training data set (Angluin, 1980). Seeking this DFA is an active research topic in grammatical inference, see (Verwer et al., 2014). The PT is iteratively made smaller by heuristically *merging* pairs of states (q, q'), and re-estimating the transition function (matrix) T. Every such merge creates a new state q'' that has the incoming and outgoing transitions of both q and q'. The merged states q and q' are removed from the model. When a merge introduces a non-deterministic choice, i.e., $T(q, a) = q_1$ and $T(q', a) = q_2$ both exist for some label a, states q_1 and q_2 are merged as well. This is called the determinization process. Which merge to perform is determined using a heuristic (typically an evidence measure). Standard EDSM, for instance, maximizes the total number of merged states with matching outputs (Lang et al., 1998). Probabilistic DFAs can be learned using statistical distances such as KL-divergence (Thollard et al., 2000) or outcomes of, for instance, likelihood ratio tests (Verwer et al., 2010b).

In DFASAT and in this chapter, the widely used *red-blue framework* (Lang et al., 1998) is applied for guiding the merge process. As shown in Figure 4.5, the red-blue framework only merges red $r \in R \subseteq Q$ and blue $b \in B \subseteq Q$ states. The red states and the transitions between them form the currently constructed DFA, the blue states are still to be identi-

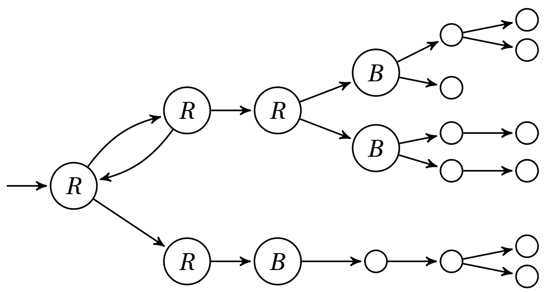


Figure 4.5: Red-Blue Framework: Starting at the root, the algorithm tries to find the smallest consistent state machine. Already identified parts of the target are marked *red*, and direct neighbors of those states as *blue*. The heuristic focuses on the fringe of the marked states, instead of having to check all possible combinations of states.

4

fied transitions, potentially to new states of the DFA. The new state q'' resulting from a red-blue merge is colored red, i.e., $R := R \cup \{q''\}$. In addition, every non-red target state $q \in Q \setminus R$ that is the target of a transition $T(r,l) = q$, for any $l \in \Sigma$, with a red source state $r \in R$, is colored blue, i.e., $B := B \cup \{q\}$. In this way, the framework maintains a core of red states with a fringe of blue states (see Figure 4.5). Initially, the start state of the APTA is colored red, and its children (targets for every symbol) are colored blue.

Merges are only allowed if the resulting DFA is still *consistent*, e.g., states with different outputs cannot be merged (Lang et al., 1998), states with significantly different outgoing transition labels cannot be merged (Heule and Verwer, 2013), or states with significantly different outgoing transition label distributions cannot be merged (Carrasco and Oncina, 1994). Overall, the run-time complexity of red-blue algorithms is bounded by $|S| \cdot n$, where S is the input set and n the size of the final model (Lang et al., 1998). For the RA learning problem, new heuristics and consistency tests are needed because the goal is to produce accurate numeric predictions instead of accurate predictions of syntactic input/output values.

MERGING FOR REGRESSION AUTOMATA

Instead of the statistical or input/output consistency checks in traditional state merging approaches described before, we allow merges between states q and q' where the mean value of difference is smaller than a given threshold. Taking the data series in Figure 4.3 for example, patterns “ab" and “bc" share a similar trend, i.e., similar difference values stored in q_1 and q_3 in Figure 4.4. We only consider merges in which all states that are merged due to determinization have sufficiently similar difference values. In addition to these difference values, we also store the number of occurrences in every state.

To evaluate possible merges and choose the best merge, we use the variant Akaike information criterion (AIC) for regression models (Burnham and Anderson, 2002) as a merge heuristic:

$$\Delta AIC = 2\left(\kappa_{before} - \kappa_{after}\right) + n \lg \frac{RSS_{before}}{RSS_{after}} \tag{4.2}$$

Algorithm 5 State-merging for Regression Automata

Require: an input sample S, an occurrence threshold t, and a difference threshold t_d
  $A = \mathsf{PT}(S)$ ▷ construct the prefix tree
  $R = \{q_0\}$ ▷ color the start state red
  $B = \{q \in Q \setminus R \mid \exists l \in \Sigma : T(q_0, l) = q\}$ ▷ color all its children blue
  while $B \neq \emptyset$ **do** ▷ while A contains blue states
    if $\exists b \in B$ s.t. $\forall r \in R$ holds $merge(A, r, b, t_d)$ = FALSE **then** ▷ if a blue state is inconsistent with all red states
      $R := R \cup \{b\}$ ▷ color b red
      $B := B \cup \{q \in Q \setminus R \mid \exists l \in \Sigma : T(q, l) = q$ and $\#\mathsf{occ}(q) \geq t\}$ ▷ color all its children with at least t occurrences blue
    else
      for all $b \in B$ and $r \in R$ **do** ▷ forall red-blue pair of states
        compute the ΔAIC of $\mathsf{merge}(A, r, b)$ ▷ find the best performing merge
      end for
      call the $\mathsf{merge}(A, r, b, t_d)$ with highest ΔAIC ▷ perform the best merge
      let q'' be resulting state
      $R := R \cup \{q''\}$ ▷ color the resulting state red
      $R := R \setminus \{r\}$ ▷ uncolor the merged red state
      $Q := Q \setminus \{r, b\}$ ▷ remove the merged states
      $B := \{q \in Q \setminus R \mid \exists r \in R, l \in \Sigma : T(q, l) = q$ and $\#\mathsf{occ}(q) \geq t\}$ ▷ recompute the set of blue states
    end if
  end while
  return A

where κ_{before} and κ_{after} are the number of parameters in the model, i.e., the number of states before and after the merge respectively, n is the number of data points in the training set for fitting the model, RSS_{before}, and RSS_{after} are the residual errors, i.e., the total square error in states before and after merge models. We compute AIC difference in each iteration of merge; there could exist more than one pair of red-blue states, i.e. candidates for merge, however, only the highest AIC difference of candidate pairs is selected for merge to improve model performance most significantly. An overview of our new state merging algorithm is given in Algorithm 5, where $\#\mathsf{occ}(q)$ denotes the number of occurrences in state q.

4.2.5. MODEL SMOOTHING

Another source of difficulty in applying syntactic models to regression tasks is model smoothing. Taking the model in Figure 4.4 for instance, it can happen that new data contain a symbol "e". For this case, no matching transition exists, and it is impossible to obtain a prediction from the model. In this chapter, we solve this problem using a relatively simple strategy: we follow the transition with the symbol closest to the input "e" according to the discretization scheme. In this example, state q_8 is reached by following the transition for symbol "d". In this way it is possible to make a numeric prediction even

Algorithm 6 Merging two regression states: merge (A, q, q', t_d)

Require: an RA $A = \langle Q, T, \Sigma, q_0, P\rangle$, two states $q, q' \in Q$, and a threshold t_d
Ensure: if q and q' are inconsistent, return FALSE; else return A with q and q' merged.
  if $|P(q) - P(q')| \geq t_d$, **then return** FALSE ▷ return FALSE if q is inconsistent with q'
  create a new state q'', and set $Q := Q \cup q''$ ▷ add a new state q'' to A
  set $\#\text{occ}(q'') := \#\text{occ}(q) + \#\text{occ}(q')$ and $P(q'') := \frac{\#\text{occ}(q)P(q)+\#\text{occ}(q')P(q')}{\#\text{occ}(q'')}$ ▷ (update $\#$occ and P)
  for all symbols $l \in \Sigma$ **do** ▷ for all transitions from q and q'
    if $T(q,l)\neg = q_\perp$ **then** set $T(q'',l) := T(q,l)$ ▷ copy outgoing transitions from q
    if $T(q,l)\neg = q_\perp$ **then** set $T(q'',l) := T(q',l)$ ▷ copy outgoing transitions from q'
  end for
  for all states $q_s \in Q$ and symbols $l \in \Sigma$ such that $T(q_s,l) \in \{q,q'\}$ **do** ▷ for all source states of transitions to q or q'
    set $T(q_s,l) := q''$ ▷ copy incoming transitions to q or q'
  end for
  for all symbols $l \in \Sigma$ **do** ▷ for all old transitions from q and q'
    if $T(q,l)\neg = q_\perp$ and $T(q',l)\neg = q_\perp$, **then** res := merge$(A', T(q,l), T(q',l), t_d)$ ▷ determinize the targets
    if res equals FALSE, **then return** FALSE and undo the merge ▷ return FALSE if the targets are inconsistent
  end for
  return true

for sequences that were neither seen in training data nor generalized to during learning. In our case studies, this only happens less than 0.1% of the time.

4.2.6. SLIDING WINDOW LENGTH

One of the key problems in our learning task is to determine the length of the sliding window, i.e., how many historical data points the prediction would rely on. Figure 4.6 illustrates the relationship between fitting error and model complexity for the wind speed training data used in the experiments. Larger length of sliding window results in more layers in PT and hence more states. E_{in} and E_{out} are the fitting mean square error in training data and testing data respectively. We can see that by increasing the model complexity (sliding window length), E_{in} decreases sharply, while E_{out} becomes increasingly worse, which is typically the result of overfitting. In practice, we favor simpler models in order to reduce the risk of overfitting. The models for which window length is less than 5, have relatively small E_{out}. We fix the length as 4 for the main experiments, and also try length 8 in order to discover whether state merging can overcome the drop in E_{out}, see Section 4.3.4.

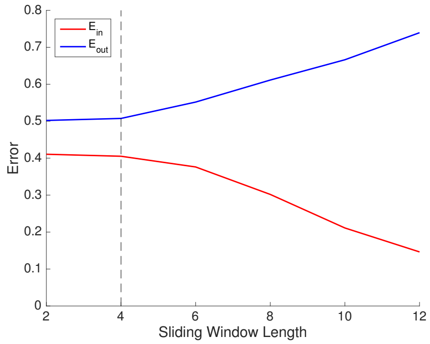


Figure 4.6: PT Fitting Error vs Window Length: Errors E_{in}, E_{out} on training data and testing data calculated on the PT, the starting data structure for the learning algorithm.

4.3. EXPERIMENTS

4.3.1. TYPICAL METHODS FOR COMPARISON

In this chapter, regression automata are compared with other widely used prediction models.

- **Persistent Model** is the most widely used baseline in time series forecasting tasks, which just let the predicted value equal its preceding known one.
- **Autoregressive Integrated Moving Average (ARIMA)** To ensure fairness when comparing prediction results, we use integrated ARMA (ARIMA) in this chapter, since we apply 1-st order derivatives in the preprocessing procedure. The maximum order of AR and MA is fixed to 3, since we have a sliding window of length 4. We select the "best fitting model" with lowest AIC and highest log-likelihood.
- **Recurrent Neural Network (RNN)** using long-term short-term nodes (Gers et al., 2001) was successful. We train a model on normalized differences input and output. We select 3 layers and 15 hidden neurons. The output function is ReLU.
- **Regression Tree (RT)** is a *IF-THEN* rules-based model, which has been applied successfully in time series forecasting (Troncoso et al., 2015). In this chapter, the regression tree is built using scikit-learn *DecisionTreeRegressor* tool,[1] which is based on the CART algorithm (Breiman et al., 1984).

4.3.2. EVALUATION METRICS

For notational convenience, we collect all the predicted data and form a new vector $\hat{\mathbf{v}} = [\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_k, \cdots, \hat{v}_N]$. The corresponding vector of actual values is defined as $\mathbf{v} = [v_1, v_2, \cdots, v_k, \cdots, v_N]$. In this chapter, the following types of indices are calculated for fair comparisons:

[1] http://scikit-learn.org/stable/modules/generated/sklearn. tree.DecisionTreeRegressor.html

- Root mean square error:

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(\hat{v}_k - v_k)^2} \tag{4.3}$$

- Mean absolute percentage error:

$$MAPE = \frac{1}{N}\sum_{k=1}^{N}\left|\frac{\hat{v}_k - v_k}{v_k}\right| \times 100\% \tag{4.4}$$

- Mean absolute error:

$$MAE = \frac{1}{N}\sum_{k=1}^{N}|\hat{v}_k - v_k| \tag{4.5}$$

4.3.3. EXPERIMENT RESULTS

CASE STUDY ONE: WIND SPEED PREDICTION

The data used in this case are from the online weather database of Delft University of Technology.[2] There are data from 16 weather stations in total. We selected station “Rijnhaven" among the stations with the longest observation period, from 2013-04-23 to 2015-10-12. We calculate hourly averages of the wind speed, and predict one hour ahead. Using a sliding window of 4 hours, the data were split into a training set containing 17537 windows with 70148 data points, and a test set containing 4113 windows with 16452 data points.

To begin with, we compare different preprocessing strategies in the prefix tree. SAX generally outperforms k-means, which provides the insights that in the wind data, the symbolization based on equal space of probability better discovers the patterns for the drift estimation. Logarithm differencing generally helps to get lower MAPE, because it reflects ratio relationship, which is consistent with the definition of MAPE. Though local SAX is powerful in anomaly pattern discovery, see (Lin et al., 2007), global SAX makes more sense in the experiment. The global SAX and differencing strategies are chosen in the following case studies. To make a fair comparison, all other baselines are fed with difference inputs.

Table 4.3: Comparisons of Different Preprocessing Strategies

Methods	Gloabl-SAX-diff	k-means-diff	Local-SAX-diff	Global-SAX-logdiff	k-means-logdiff	Local-SAX-logdiff
RMSE (m/s)	**0.5031**	0.6501	0.5115	0.5072	0.6211	0.5124
MAPE (%)	18.7711	25.3068	18.9490	**18.3330**	20.6989	18.7300
MAE (m/s)	**0.3660**	0.4850	0.3725	0.3666	0.4347	0.3722

The evaluation results of different models are summarized in Table 4.4, where the best for each index is in bold. Our model outperforms all other baselines in MAPE while ARIMA shows slightly better results in RMSE and MAE.

[2]http://weather.tudelft.nl/csv/

The final merged state machine is illustrated in Figure 4.7. The model's size is drastically reduced from 350 states to 20 states (except the start state and the leaf states, since they are useless for the regression). The top-most state is the start state. Starting from this state, the model moves along transitions by first discretizing the next time series value and then following the transition with that discretized label. The first value in every state (a circle) is the mean value of difference values from the training data reaching that state. These are used to make predictions. The second value shows the number of occurrences of every state.

The automaton has 11 loops, i.e., transitions where origin and target state are the same, which are introduced by state merge. Given the historic data that are already abstracted into the pattern *abc* (continuously increasing wind speed) for instance, it starts from root state and reaches the state 0.054, which means it is expected to drift up 0.054 m/s. And for the pattern *hgf*, it reaches the state -0.107, which predicts a 0.107 m/s drift down. The pattern *hhh* staying high speed for 3 hours, reaches -0.145 and is predicted to slope down.

Table 4.4: One-hour-ahead Speed Prediction Performance Comparisons.

Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
RMSE (m/s)	0.4996	0.5031	**0.4956**	0.6060	0.6884	0.5077
MAPE (%)	**18.5797**	18.7711	18.7355	24.483	27.1475	18.6090
MAE (m/s)	0.3629	0.3660	**0.3615**	0.4707	0.5116	0.3685

Table 4.5: 3-hour-ahead Speed Prediction Performance Comparisons.

Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
RMSE (m/s)	**0.8722**	0.8753	0.8821	1.0015	0.9892	0.8930
MAPE (%)	**32.5249**	32.6794	33.1649	37.2406	38.8493	33.2933
MAE (m/s)	**0.6321**	0.6347	0.6432	0.7637	0.7404	0.6489

Table 4.6: 6-hour-ahead Speed Prediction Performance Comparisons.

Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
RMSE (m/s)	**1.2048**	1.2083	1.2286	1.2617	1.3038	1.2344
MAPE (%)	**46.8085**	47.0155	48.0161	47.02642	51.9327	48.1143
MAE (m/s)	**0.8974**	0.9013	0.9192	0.9444	0.9855	0.9226

CASE STUDY TWO: MULTI-STEP PREDICTION

In this case study, we evaluate the regression models for multi-steps, i.e., more than one-hour-ahead forecasting, still using the data sets with one data point per hour. Our input data again consist of windows, pre-processed as in the previous case studies, except for the last element of the window being the value for multiple steps ahead. For

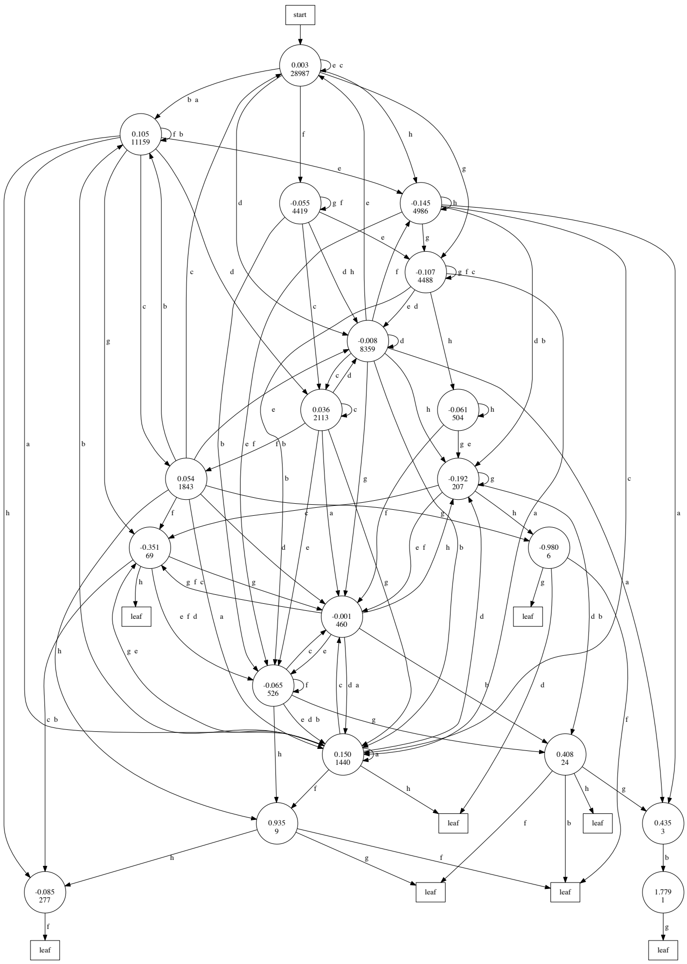


Figure 4.7: The merged RA for the one-hour-ahead wind-speed prediction.

example, to predict a value three hours into the future, at time $T+3$, our training data contain windows of the form $(x_{T-2}, x_{T-1}, x_T, x_{T+3})$. The evaluation results of 3-hour-ahead and 6-hour-ahead predictions are listed in Table 4.5 and Table 4.6. With the increasing of prediction interval, the persistent model doesn't work as well as in *Case One*. Our model improves significantly compared to other approaches.

CASE STUDY THREE: WIND POWER PREDICTION

In this case study, we investigate wind power prediction using the data set from the National Renewable Energy Laboratory (NREL) of U.S. Department of Energy.[3] The train-

[3] http://www.nrel.gov/electricity/transmission/eastern_wind _dataset.html

ing data start from *2004-01-02 00:00:00* to *2006-05-31 23:50:00*, while the testing data start from *2006-06-01 00:00:00* to *2007-01-01 23:50:00*.

Similar to the wind speed forecasting case study, we apply our model to wind power prediction, i.e., using the historical wind power data as input and the one, three, and six hour ahead power as output. Wind power forecasting is challenging due to the non-linearity resulting from the dead zone and the saturation characteristics. More specifically, power output has zero value when the wind speed value is lower than the wind turbines' cut-in threshold; meanwhile, the output reaches constant rated power if the wind speed is greater than the cut-off upper-bound. Table 4.7 gives a comparison of the power prediction for different models. Note that due to the dead zone characteristic of wind power system, many zero value exists in the real data making the MAPE metric ill-defined. Only RMSE and MAE are reported for comparison. From the results we can see that the ARIMA performance is better in the 1-hour-ahead data set. ARIMA is powerful in one step ahead because the on-line updating of both input autoregressive values and residual errors is efficient in short-term forecasting. However, in relatively longer prediction intervals, our model gains improvement over baselines.

Table 4.7: Power Prediction Performance Comparisons.

	Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
1-hour-ahead	RMSE (MW)	1.8952	1.8979	**1.8673**	1.9859	2.6541	1.9830
	MAE (MW)	1.2610	1.2613	**1.2312**	1.2814	1.8066	1.2793
3-hour-ahead	RMSE (MW)	**3.7427**	3.7435	3.7738	4.6883	4.4193	3.8796
	MAE (MW)	**2.6438**	2.6458	2.6196	3.6595	3.1597	2.6832
6-hour-ahead	RMSE (MW)	**5.0053**	5.0088	5.0434	5.1567	5.4872	5.1486
	MAE (MW)	**3.6529**	3.6546	3.6540	3.7355	4.0661	3.6529

4.3.4. LEARNING AND MODEL COMPLEXITY

Learning finite state automata exactly with incomplete samples is NP-hard (Angluin, 1978). State-merging algorithms use heuristics, and generally have a worst-case complexity on the order of a cubic term in the input data size. Evaluating a regression automaton is a linear sequence of looking up the transitions to the last node, and adding the predicted speed difference to the previous speed value. Our automata only have about 20 states, requiring storing 20 float values and at most $20 \times |\Sigma|$ triples of state-symbol-state for the transition matrix. In practice, the runtime of RAs, including training and testing, on our Intel 2.6 GHz i5 processors using a single core doesn't need more than a minute. The comparisons with all baselines are listed in Table 4.9. We also compare the performance of the prefix tree with the performance our merged regression automata. The prefix tree is a compact representation of the input data and is generated in linear time. While it is generated much faster, it does not generalize, and is large in size. Figure 4.6 shows the training and testing error in prefix trees with different depths. The longer the window size, i.e. the higher the order of auto-regression, the deeper the prefix tree will get. We try to investigate how state merging influences the model performance and how it relates to varying size measured in states. Table 4.8 shows the benefit of the learning process. $impr(\%)$ is the automaton's improvement in RMSE over prefix trees. For longer sliding windows, state merging clearly improves the RA's performance more. The RMSE of the model with length-8 has accuracy very close to the length-4 model after learning. It surprisingly provides evidence for the generalization efficiency of our learning algorithm.

Table 4.8: Improvement due to state-merging over the prefix tree in the RSME measure at different sliding window length.

	1-hour-ahead			3-hour-ahead			6-hour-ahead		
	RA	Prefix Tree	impr (%)	RA	Prefix Tree	impr (%)	RA	Prefix Tree	impr (%)
length-4	0.4996	0.5031	0.70	0.8722	0.8753	0.35	1.2048	1.2083	0.29
length-8	0.4994	0.5959	16.19	0.8737	0.9333	6.39	1.2089	1.2495	3.25

Table 4.9: Runtime Comparisons.

Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
Runtime	19.086s	1.806s	1m48.796s	19m54.580s	2.035s	1.081s

4.4. CONCLUSION

The main contribution of this work is the extension of automata for time series regression. A novel state merging approach for learning small automata from numeric data is proposed using the DFASAT framework. To the best of our knowledge, we provide the first automaton model together with a learning algorithm that can be directly applied to time series regression problems. Several case studies are performed, which demonstrate that our approach allows for powerful generalization from training to testing data. In addition to good performance in practice, our algorithm provides succinct and interpretable models, which can be essential for deployment in real wind power parks. In the near future, we will make even more use of the numeric wind speed/power data during merging. This way, we can exploit spatial information, either by modifying our preprocessing to create a multivariate regression problem, or considering additional information such as location, directionality, correlation, and standard deviations during consistency checks and merging. Additionally, different discretization strategies could be further investigated for better abstraction of numerical data. An interesting approach would be to discretize these data on-the-fly during the learning process, as has been done before with temporal data in timed automata (Verwer et al., 2010b). In addition to mean forecasting, probabilistic prediction is also important for decision purposes (Pinson et al., 2013). RAs can generate a probabilistic forecasting, which will be done in the future. We will also try the rolling evaluation for concept drift problems (Giles et al., 2001).

5

LEARNING AUTOMATA FOR PERCEPTION AND CONTROL

From this chapter, we start the discussion about the safety problem in an intelligent control system. We applied stochastic automata learning for profiling the lane change behaviors of human drivers. The lane change intention is modeled as a stochastic input to a car-following controller of an ego-vehicle. The experiments demonstrate the enhanced safety by predicting such intentions.

The material in this chapter has appeared in
Yihuan Zhang, Qin Lin, Jun Wang, Sicco Verwer, and John Dolan. Lane-change intention estimation for car-following control in autonomous driving. IEEE Transactions on Intelligent Vehicles, 3(3): 276–286, 2018

5.1. INTRODUCTION

Recently, many research institutes and vehicle manufacturers have focused on the commercialization of autonomous driving systems. Safety and reliability are fundamental for self-driving cars on roads. Most car crashes are caused by human mistakes, and many of these occur during lane changes (Lum and Reagan, 1995; Peden et al., 2004). Furthermore, fewer than 50% of drivers use turn signals when they change lanes (Dang et al., 2013). In order to guarantee the safety of driving, it is important for self-driving cars to estimate the driving behavior of surrounding vehicles and predict their intention of lane change before they cross lane lines.

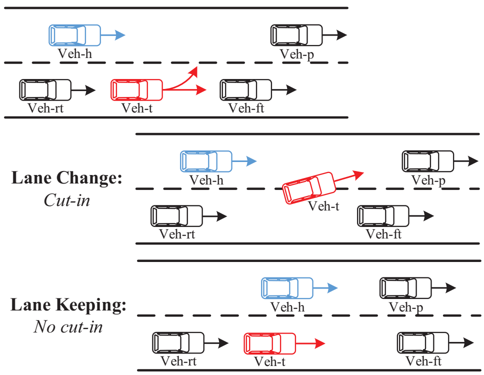


Figure 5.1: Multi-lane car-following scenarios.

Figure 5.1 illustrates the scenarios in highway driving. The self-driving car is noted as the host vehicle in blue (*Veh-h*), the target vehicle is in red (*Veh-t*), the proceeding vehicle (*Veh-p*) is in front of the host vehicle, *Veh-ft* and *Veh-rt* represent the front and rear vehicles in the target lane. Assume that the red vehicle is following the leading vehicle and intends to merge. In this case, if the host vehicle cannot estimate the merge intention of the red vehicle, a sudden change of acceleration may occur, which leads to an uncomfortable or even dangerous situation. Human drivers predict the behavior of surrounding vehicles (merging into their lane or not) based on their observations and driving experiences. A self-driving car uses a computational model to mimic human beings and estimate the states of its own and surrounding vehicles.

The cut-in intention of the target vehicle should be estimated to ensure a safe and comfortable car-following for the host vehicle. The contextual information of the four surrounding vehicles is used to model the driving behavior of the target vehicle. In this chapter, we need to recognize/classify the observations (vehicle positions, lateral accelerations, etc.) into lane change or lane keeping. It is a standard multivariate-time-series classification based on the observations, i.e., to assign a label to a complete sequence of lane change or lane keeping. This work aims at an even more challenging task of predicting such a label (i.e., intention) in advance for the intervention of control.

Although Adaptive Cruise Control (ACC) systems have been on the market since 1995 (Rajamani, 2015), their performance in terms of smoothness is frequently interrupted by cut-in vehicles from adjacent lanes. More attention should be paid to the intention of other vehicles for a more reliable ACC. In this chapter, an intention-based car-following control method is proposed by integrating the cut-in intention of surrounding vehicles.

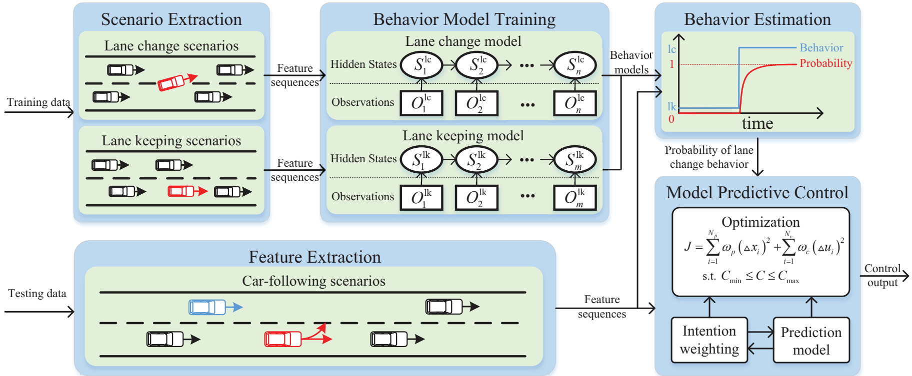


Figure 5.2: Framework of proposed method.

The framework of the proposed method is shown in Figure 5.2. First, a scenario extraction method is used to obtain two classes of driving sequences: lane change and lane keeping. Then, the continuous Hidden Markov Models (HMMs) integrated with the Gaussian Mixture Models (GMMs) are used to model the behavior of lane change and lane keeping, respectively. A likelihood function is employed to estimate the behavior in an online manner. Finally, a framework of model predictive control is proposed to consider the predicted cut-in intention.

The major contributions of this chapter are as follows:

- To the best of our knowledge, this is the first work to fuse traffic contextual information into the driving behavior estimation of target vehicles by using continuous HMMs.
- A threshold-based method is used to estimate driving behavior of a target vehicle in a streaming fashion, which is able to predict the behavior of lane change before the target vehicle crosses the lane line.
- A novel car-following control method integrating the cut-in intention estimation is proposed and achieves superior performance in terms of comfort and safety.

The remainder of this chapter is organized as follows. Related work is introduced in Section 5.2. The proposed method is detailed in Section 5.3. The experiments are carried out in Section 5.4. Conclusions and future work are presented in Section 5.5.

5.2. RELATED WORK

The related work is divided into two parts: one is on the estimation and prediction of driving behavior by using various kinds of information, the other is on car-following control including mathematical models, control methods and ACC systems.

5.2.1. DRIVING BEHAVIOR CLASSIFICATION

Many researches focused on the classification and prediction of driving behavior. In (Meyer-Delius et al., 2009), the behavior of following and passing a vehicle was modeled and recognized using HMMs and Gaussian mixture model. In (Schreier et al., 2016), a maneuver-based method was proposed to estimate the driving state of a driver and to predict the future trajectory considering the information of its leading vehicle. In car-following scenarios, it is important to monitor the situation in the adjacent lanes to deal with the behavior of lane change.

The behavior estimation or intention recognition of lane change can be classified into two categories based on its input signals. The first one uses ***internal*** information of a target vehicle such as throttle pedal pressure, brake pedal pressure and steering wheel angles to identify driving behavior. It is mainly used in advanced driver assistance systems. In (Pentland and Liu, 1999), an accuracy of 93.3% over the 47 recorded lane-change scenarios was achieved based on the data of vehicle accelerations, brakings and steerings. In (Hou et al., 2011), lateral accelerations, steering wheel angles and steering angles were used to classify the maneuvers of lane keeping and lane change by continuous HMMs and the average recognition rate of lane change was over 90%. In (Li et al., 2015), lane change maneuvers were recognized by using the features extracted from vehicle states and driver operation signals. The dataset was recorded from different drivers under varying driving conditions and the recognition rate was 88.2%. In (Doshi and Trivedi, 2009), some additional features like eye movements and head dynamics were added to the behavior recognition for improved accuracy. In (Wang et al., 2016), the signals of heart electrocardiogram, galvanic skin responses and respiration were utilized to train a multi-layer neural-network model. The prediction of lane change was achieved about 2 seconds before the target vehicle actually crossed lane lines.

The other category uses ***external*** information of a target vehicle, for example, vehicle speeds, lateral offsets, distances, etc. It is possible for self-driving cars to estimate the behavior of surrounding vehicles because all parameters are measurable by sensors on board. In (Kumar et al., 2013), lateral positions and relative heading angles were used as features to train a support vector machine (SVM) and a Bayesian filter was used to obtain the probability of driving behaviors. However, the effect of surrounding vehicles on the behavior of the target vehicle should not be ignored. More and more researchers have considered the surrounding traffic when studying driving behaviors. In (Morris et al., 2011), the lane change intention was estimated based on the driver's head motions, internal signals and the information of the surrounding vehicles. The classifier was able to provide the intention of the driver more accurately.

The dataset of Next Generation SIMulation (NGSIM) has been adopted to explore the characteristics of the vigilant lane-change process. In (Balal et al., 2016), a fuzzy inference system was used to make a decision of lane change based on distances and relative speeds. In (Bi et al., 2016), a neural-network based learning method was applied to model the behavior of lane change. The SVM-based classification as a classical machine learning method can deal with high-dimensional input features. In (Nie et al., 2016; Woo et al., 2016), SVMs were used to classify different situations of lane-change behavior, and different input features of surrounding vehicles were used to train the SVMs. In addition, a probabilistic classification method based on a Bayesian network was applied in (Yan et al., 2016; Rehder et al., 2016). The time-to-collision between a target vehicle and surrounding vehicles was used as an input feature to obtain the probability of lane-change behavior. An exponential probability model of lane-change was proposed in (Lee et al., 2016) by using NGSIM data. Various factors were claimed to affect the decision of lane change, including the relative speeds between the target and original lanes and the distances between the target vehicle and the surrounding ones.

5.2.2. CAR-FOLLOWING CONTROL

The first work on car-following can be dated back to the 1950s. In (Pipes, 1953), a linear follow-the-leader model was proposed to calculate the desired acceleration by using the relative speeds between the following and the leading vehicles. Another widely-used linear model, knows as the Helly model, was proposed in (Helly, 1959b). Alternatively, a non-linear Gazis-Herman-Rothery model introduced the power operators of ranges and speeds (Gazis et al., 1961). An intelligent driver model was introduced in (Treiber et al., 2000) to simulate freeway and urban traffic. In our recent work (Zhang et al., 2017a), a human-like car-following controller was designed to mimic human driving behavior. These works are essentially feed-forward models that are more suitable for simulating car-following behavior than real-time control.

The ACC system as an upgrade of cruise control improves the convenience and safety of driving. Many control methods have been applied to ACC systems, e.g., proportional-integral (PI) control (Rajamani, 2011), fuzzy control (Sathiyan et al., 2015), and model predictive control (MPC) (Schmied et al., 2015; Kamal et al., 2015). The MPC method can be used to deal with multiple objective optimizations of driving safety, fuel efficiency and ride comfort. In (Schmied et al., 2016), a scenario MPC method was proposed that enabled predictive and anticipatory driving in multi-lane and multi-vehicle scenarios. By using a stochastic modeling approach, the lane-change probability of surrounding traffic participants was determined and integrated into the optimization. Simulations illustrated the much smoother control of speeds and accelerations than PI control. In (Liu et al., 2017a), a car-following gap model was generated from the data of highway naturalistic driving, and the cut-in probability was incorporated into the algorithm of MPC control. Simulated scenarios demonstrated the smoothness of vehicle driving. Although these methods have considered the behavior of vehicles in adjacent lanes, the methods of intention estimation were only tested by simulated data rather than real traffic data. In this work, the models of driving behavior are learned from the real data, and all the tests are conducted in real driving scenarios.

In summary, the smooth and reliable performance of ACC systems tends to be interrupted by cut-in vehicles from adjacent lanes. A model of behavior estimation is crucial for improving the performance of ACC systems. This chapter focuses on predicting the cut-in intention at “any time" (i.e., an online fashion) from the external information of surrounding vehicles. The inferred cut-in probability is integrated into the framework of MPC control to efficiently deal with the sudden behavior change of target vehicles.

5.3. PROPOSED METHOD

In the NGSIM dataset, separated scenarios for each vehicle are extracted where surrounding vehicles remain the same. Two types of behavior models, i.e., lane keeping and lane change, are learned using GMM-HMMs. In the testing phase, the likelihood of sequences is computed using a forward algorithm and is compared with a threshold for the final recognition. The probability of lane-change is calculated and integrated into the MPC framework to control the car-following behavior of the host vehicle.

5.3.1. SCENARIO DEFINITION AND EXTRACTION

In the folowing, the NGSIM dataset is described in detail and the scenarios used in this chapter are defined.

DATA DESCRIPTION

This chapter uses the public dataset NGSIM (NGSIM, 2007), a program funded by the U.S. Federal Highway Administration. These trajectory data are thus far unique in the history of traffic research and provide a valuable basis for the research of driving behavior on structured roads. All the experiments are performed on the datasets of I-80 and US-101. The labeled scenario data are open-sourced.[1]

The I-80 dataset consists of three 15-minute periods: 4:00 pm to 4:15 pm, 5:00 pm to 5:15 pm, and 5:15 pm to 5:30 pm. These periods represent respectively a buildup of congestion, a transition between uncongested and congested conditions, and full congestion. A total of 45 minutes of data are available in the US-101 dataset, which are segmented into three 15-minute periods: 7:50 am to 8:05 am, 8:05 am to 8:20 am, and 8:20 am to 8:35 am. The vehicle trajectories in both datasets data include the precise location of each vehicle within the study area and the data were sampled at a rate of 10 Hz.

SCENARIO SEGMENTATION

The segmented scenarios in Figure 5.1 have the following properties:

- In each scenario, the surrounding vehicles (*Veh-h*, *Veh-p*, *Veh-ft*, *Veh-rt*) of a target vehicle (*Veh-t*) remain the same.
- We set the relative distance to 150 m and the relative speed to 0 for any missing surrounding vehicles.
- A scenario ends when a target vehicle crosses a lane line (merge), passes *Veh-p*, or yields to *Veh-h*.
- A new scenario restarts immediately once the preceding scenario is finished to ensure continuity between driving scenarios.
- The segmented scenarios last at least two seconds to ensure complete lane-change or lane-keeping behavior.

[1] All the labeled scenario data can be found in our online repository: https://bitbucket.org/stzyhian/beta-ngsim.

Table 5.1: Scenario segmentations.

Dataset	Lane change	Lane keeping
I-80-1	212 (avg. dur. 6.12s)	16997 (avg. dur. 6.01s)
I-80-2	159 (avg. dur. 6.13s)	16972 (avg. dur. 6.16s)
I-80-3	167 (avg. dur. 6.30s)	16536 (avg. dur. 6.26s)
US-101-1	242 (avg. dur. 8.07s)	15683 (avg. dur. 7.99s)
US-101-2	156 (avg. dur. 8.56s)	17254 (avg. dur. 8.07s)
US-101-3	154 (avg. dur. 7.44s)	17796 (avg. dur. 7.71s)

The summary of the segmented sequences in both datasets is shown in Table 5.1. The average duration of each scenario segmentation is about 6 to 8 seconds. The highly imbalanced data, i.e., much higher proportion of lane keeping than lane change, pose another significant challenge to behavior recognition. However, the proportion of data is consistent with daily driving. According to References (Balal et al., 2016; Bi et al., 2016), the features listed in Table 5.2 are deemed relevant and are extracted.

Table 5.2: Features of scenario segmentation.

Symbols	Descriptions
v_{x}	Longitudinal speed of *Veh-t*
d_{o}	Lateral speed of *Veh-t*
d_{o}	Lateral offset from target lane line to *Veh-t*
$\Delta v_{\mathrm{t,p}}$	Longitudinal speed difference between *Veh-t* and *Veh-p*
$\Delta v_{\mathrm{t,h}}$	Longitudinal speed difference between *Veh-t* and *Veh-h*
$\Delta v_{\mathrm{t,ft}}$	Longitudinal speed difference between *Veh-t* and *Veh-ft*
$\Delta v_{\mathrm{t,rt}}$	Longitudinal speed difference between *Veh-t* and *Veh-rt*
$\Delta x_{\mathrm{t,p}}$	Longitudinal distance between *Veh-t* and *Veh-p*
$\Delta x_{\mathrm{t,h}}$	Longitudinal distance between *Veh-t* and *Veh-h*
$\Delta x_{\mathrm{t,ft}}$	Longitudinal distance between *Veh-t* and *Veh-ft*
$\Delta x_{\mathrm{t,rt}}$	Longitudinal distance between *Veh-t* and *Veh-rt*

5.3.2. BEHAVIOR MODEL

HMMs have been widely used to model driving behavior due to their powerful ability to describe dynamic processes and infer unobserved (hidden) states (Tang et al., 2016; Meyer-Delius et al., 2009). GMMs are used to model the probabilities of the continuous observations such as speeds.

GMM

The variables in Table 5.2 can be classified into three categories as follows:

$$\begin{aligned}\xi_t = \Big[& \left[v_{\mathrm{x}}(t), v_{\mathrm{y}}(t), d_{\mathrm{o}}(t)\right], \\ & \left[\Delta v_{\mathrm{t,p}}(t), \Delta v_{\mathrm{t,h}}(t), \Delta x_{\mathrm{t,p}}(t), \Delta x_{\mathrm{t,h}}(t)\right], \\ & \left[\Delta v_{\mathrm{t,ft}}(t), \Delta v_{\mathrm{t,rt}}(t), \Delta x_{\mathrm{t,ft}}(t), \Delta x_{\mathrm{t,rt}}(t)\right] \Big]^{\mathrm{T}}\end{aligned}$$

ξ_t is used to model the behaviors, and the first group $\left[v_x(t), v_y(t), d_o(t)\right]^{\mathrm{T}}$ is used to build the model which only considers the information of target vehicles. In this chapter, we assume that the distribution of the observation ξ is a weighted sum of multivariate Gaussian distribution functions:

$$
\begin{aligned}
p(\xi_t;\theta) &= \sum_{k=1}^{K} \omega_k \mathcal{N}(\xi_t;\mu_k,\Sigma_k) \\
&= \sum_{k=1}^{K} \frac{\omega_k \cdot \exp\left(-\frac{1}{2}(\xi_t-\mu_k)^{\mathrm{T}}\Sigma_k^{-1}(\xi_t-\mu_k)\right)}{\sqrt{(2\pi)^{11}\det(\Sigma_k)}}
\end{aligned}
\tag{5.1}
$$

where $\theta = \{\theta_k\}_{k=1}^{K} = \{\omega_k,\mu_k,\Sigma_k\}_{k=1}^{K}$ are the parameters of the GMMs, $\mathcal{N}(\xi_t;\mu_k,\Sigma_k)$ is the multivariate Gaussian distribution with the mean center $\mu_k \in \mathscr{R}^{11\times 1}$ and covariance matrix $\Sigma_k \in \mathscr{R}^{11\times 11}$, and K is the number of GMM components which can be determined using the Bayesian information criterion (BIC) (Findley, 1991). As $\omega_k \in (0,1]$ is the weight of the k^{th} Gaussian component, we have $\sum_{k=1}^{K}\omega_k = 1$.

Given a data sequence $\xi_{1:n}$, the maximum-likelihood estimation method is used to find a θ that maximizes the likelihood of the GMM function:

$$
\mathscr{L}(\theta) = \sum_{t=1}^{n} \ln(p(\xi_t;\theta))
\tag{5.2}
$$

The expectation-maximization algorithm is utilized in this chapter to search for the optimal parameter

$$
\theta^* = \arg\max_{\theta} \mathscr{L}(\theta)
$$

The estimation of θ at Step j is denoted by $\hat{\theta}^j$. The iteration from $\hat{\theta}^j$ to $\hat{\theta}^{j+1}$ is achieved by the following *E-step* and *M-step* (Bilmes et al., 1998).

- *E-step*: For each iteration, the posterior probability for each component k is calculated by using the previous estimation $\hat{\theta}^j$:

$$
P_k^{j+1}(\xi_t) = \frac{\hat{\omega}_k^j \cdot \mathcal{N}(\xi_t;\hat{\mu}_k^j,\hat{\Sigma}_k^j)}{\sum_{l=1}^{K}\hat{\omega}_l^j \cdot \mathcal{N}(\xi_t;\hat{\mu}_l^j,\hat{\Sigma}_l^j)}
\tag{5.3}
$$

- *M-step*: The model parameters are then updated by

$$
\begin{aligned}
\hat{\omega}_k^{j+1} &= \frac{1}{n}\sum_{t=1}^{n} P_k^{j+1}(\xi_t) \\
\hat{\mu}_k^{j+1} &= \frac{\sum_{t=1}^{n}(\xi_t \cdot P_k^{j+1}(\xi_t))}{\sum_{t=1}^{n} P_k^{j+1}(\xi_t)} \\
\hat{\Sigma}_k^{j+1} &= \frac{\sum_{t=1}^{n}\left(P_k^{j+1}(\xi_t)(\xi_t-\hat{\mu}_k^{j+1})(\xi_t-\hat{\mu}_k^{j+1})^{\mathrm{T}}\right)}{\sum_{t=1}^{n} P_k^{j+1}(\xi_t)}
\end{aligned}
$$

At the end of each iteration, the log-likelihood $\mathscr{L}(\hat{\theta}^{j+1})$ is calculated by

$$\mathscr{L}(\hat{\theta}^{j+1}) = \sum_{t=1}^{n} \mathscr{L}(\hat{\theta}^{j}) \tag{5.4}$$

The iteration will continue until the likelihood difference between two consecutive estimated models is less than a threshold, which is set to 10^{-10} here.

HMM

Two separate HMMs are built to represent the behavior of lane change and lane keeping. In this chapter, the structure of the HMM is left-to-right, as shown in Figure 6.1 (behavior model training block). The HMM is represented by

$$\lambda = \{\mathscr{S}, \mathscr{Z}, \mathscr{A}, \mathscr{B}, \pi\}$$

where

- $\mathscr{S} = \{s_1, \cdots, s_N\}$ represents a finite set of N hidden states.
- $\mathscr{Z} = \{\xi_t\}$ is the set of all observed states ξ at time t and each ξ consists of the eleven elements included in the GMM.
- $\mathscr{A} = [a_{ij}]$ is the state transition matrix and a_{ij} is defined as the probability of a transition from state s_i to state s_j.
- $\mathscr{B} = \{b_i(\xi)\}$ is the observation model and $b_i(\xi)$ represents the probability of observing ξ while being in state s_i.
- $\pi = \{\pi_i\}$ is the initial state distribution where π_i represents the probability of the state s_i being the initial state.

Readers can refer to (Rabiner, 1989a) for a more detailed formulation and applications of HMM. HMM is a dual stochastic model: one is a Markov model for stochastic state transition, the other is the stochastic observation in each state. Three hidden states are chosen to represent the underlying dynamic processes of the lane-change and lane-keeping behavior. The continuous observation model $\mathscr{B}$ is defined by

$$b_i(\xi) = \sum_{k=1}^{K} \omega_k \mathscr{N}(\xi; \mu_k, \Sigma_k) \tag{5.5}$$

The Baum-Welch algorithm (Dempster et al., 1977) is used to estimate λ of the two HMMs. It is an approximate iterative optimization technique for maximizing the likelihood of the observations. A random set of initial parameters is chosen and improved by gradient updating.

BEHAVIOR RECOGNITION

In the testing phase, a binary recognition, i.e., lane change or lane keeping, is achieved in a receding horizon manner. Assume that the sequence $\xi_{1:n}$ is a complete period of lane change/keeping, where n is the length of the sequence. The shortest sequence with a size s implies the least information to distinguish two kinds of behavior. A prediction can be achieved if $s < n$. The streaming data $\xi_{1:t}$ where $t \geq s$ are fed as the real-time input to λ_{lk} and λ_{lc} separately for likelihood computation. λ_{lk} and λ_{lc} respectively represent the HMM of lane keeping and lane change. $P(\xi_{1:t}|\lambda_i)$ is obtained by a forward algorithm (Rabiner, 1989a):

$$P(\xi_{1:t}|\lambda_i) = \sum_{i=1}^{N} \alpha_t(i) \tag{5.6}$$

where

$$\begin{aligned} \alpha_{t+1}(j) &= \left(\sum_{i=1}^{N} \alpha_t(i) \cdot a_{ij} \right) b_j(\xi_{t+1}) \\ \alpha_1(j) &= \pi_j b_j(\xi_1) \end{aligned} \tag{5.7}$$

As there is no prior knowledge of the driving behavior of a specific driver, we assume the prior probabilities of each model are identical. After the calculation of $P(\xi_{1:t}|\lambda_{\text{lk}})$ and $P(\xi_{1:t}|\lambda_{\text{lc}})$, we are able to set a threshold to estimate the current behavior of the target vehicle:

$$\mathscr{R} = \frac{P(\xi_{1:t}|\lambda_{\text{lc}})}{P(\xi_{1:t}|\lambda_{\text{lk}})} \tag{5.8}$$

where $\mathscr{R}$ indicates whether the classification is more likely to be lane change or lane keeping.

5.3.3. MODEL PREDICTIVE CONTROL

Once the behavior model is built, a probability of lane change is calculated and integrated into the framework of model predictive control.

INTENTION ESTIMATION

The probability of the lane change intention is calculated as follows:

$$P_{\text{c}} = \begin{cases} \tanh\left(\omega_{\text{c}} \cdot \dfrac{\mathscr{R} - \mathscr{R}_{\text{T}}}{\mathscr{R}_{\text{m}} - \mathscr{R}_{\text{T}}}\right), & \mathscr{R} > \mathscr{R}_{\text{T}} \\ 0, & \mathscr{R} \leq \mathscr{R}_{\text{T}} \end{cases} \tag{5.9}$$

where $\mathscr{R}_{\text{T}}$ is the threshold of the classification, $\mathscr{R}_{\text{m}}$ is the maximum ratio obtained from the training data and ω_{c} is a span parameter indicating the range of the ratio. The likelihood is thus normalized as a probability ranging from 0 to 1. The function "tanh" is selected because the values of random variable $\frac{\mathscr{R}-\mathscr{R}_{\text{T}}}{\mathscr{R}_{\text{m}}-\mathscr{R}_{\text{T}}}$ in the training dataset follow such a distribution with the smallest fitting error.

PREDICTION MODEL

In this chapter, the longitudinal motion of the vehicle is expressed by

$$\begin{aligned} x(t+1) &= x(t) + v(t)\Delta t + 0.5a(t)\Delta t^2 \\ v(t+1) &= v(t) + a(t)\Delta t \end{aligned} \tag{5.10}$$

where x, v, a are respectively the positions, speeds and accelerations of the host vehicle, and Δt is the sampling time. Then the following variables are defined:

- Distances: $\Delta x = x_\mathrm{f} - x_\mathrm{h}$ where x_f is the longitudinal position of the virtual leading vehicle, and $x_\mathrm{f} = P_\mathrm{c} x_\mathrm{t} + (1 - P_\mathrm{c}) x_\mathrm{p}$. Note that, if the probability P_c is 1, then the host vehicle will assume the target vehicle to be the leading vehicle.
- Relative speeds: $\Delta v = v_\mathrm{f} - v_\mathrm{h}$ where v_f is the longitudinal speed of the virtual leading vehicle, and $v_\mathrm{f} = P_\mathrm{c} v_\mathrm{t} + (1 - P_\mathrm{c}) v_\mathrm{p}$.
- Accelerations: $a_\mathrm{h} = j_\mathrm{h} \Delta t$ where j_h is the jerk of the host vehicle.

Due to the uncertainty of the vehicle motions, we assume that the accelerations of the surrounding vehicles remain the same in the prediction step as in Reference (Liu et al., 2017a). Such an assumption is reasonable because the prediction window of the MPC is continuously receding to the next time point when the real status of the leading vehicles is updated.

RECEDING HORIZON OPTIMIZATION

The cost function of the MPC is designed to meet the following objectives:

- Tracking errors: The objective of car-following control is to follow the speed of the leading vehicle while keeping a safe distance. The distance is defined as a constant time headway policy (Schmied et al., 2016):

  $$d_\mathrm{des} = d_0 + \tau_\mathrm{h1} v_\mathrm{h} + \tau_\mathrm{h2} \Delta v$$

  where d_0 denotes the desired distance at standstill, and τ_h1, τ_h2 are constant-time headway parameters.

  $$J_\mathrm{T} = \omega_\mathrm{d} (d_\mathrm{des} - \Delta x)^2 + \omega_\mathrm{v} \Delta v^2 \tag{5.11}$$

- Comfort and smoothness: The host vehicle should realize a comfortable and economic driving style by minimizing its accelerations and jerks.

  $$J_\mathrm{C} = \omega_\mathrm{a} a_h^2 + \omega_\mathrm{j} j_h^2 \tag{5.12}$$

  where ω_d, ω_v, ω_a and ω_u are the weight values of the cost function.

Considering the nonholonomic constraints of the vehicle and the car-following scenario, the following constraints should also be considered in the MPC design:

- The speed of the host vehicle is bounded by

  $$0 \le v_\mathrm{h} \le v_\mathrm{max}$$

- The minimum gap from the leading vehicle is constrained by

$$d_{\text{safe}} \le \Delta x$$

where $d_{\text{safe}} = \tau_0 v_{\text{h}}$ is the minimum time headway.

- The acceleration constraint of the host vehicle is

$$a_{\min} \le a_{\text{h}} \le a_{\max}$$

- The jerk constraint of the host vehicle is

$$j_{\min} \le j_{\text{h}} \le j_{\max}$$

5

The optimization problem can now be written as:

$$\min_{j_{\text{h}}(k),k=1,\cdots,N_{\text{p}}} J = J_{\text{T}} + J_{\text{C}} \tag{5.13}$$

where N_{p} is the prediction step. The optimization problem is subject to the above constraints. Note that the optimal solution is a vector of control values with the length N_{p}. The MPC method only takes the first value and then moves to the next time point and re-starts the optimization.

5.4. EXPERIMENTAL RESULTS

The effectiveness of the proposed method is demonstrated by a 5-fold cross-validation experiment. In order to balance the data proportion of lane change and keeping, an equal number of data, i.e., 538 sequences, are randomly chosen. First, the BICs are calculated to determine the number of GMM components, where n_t is the length of training data and $\hat{\mathscr{L}}$ is the maximum log-likelihood. When fitting GMMs, it is possible to increase the likelihood by increasing K, which may result in over-fitting. Moreover, the log-likelihood may have a large negative value and the two parts in this equation may not be of the same order of magnitude. A normalization step is taken to make a trade-off between number of parameters and log-likelihood:

$$\widehat{\text{BIC}} = \ln(n_t) \cdot \frac{K - K_{\min}}{K_{\max} - K_{\min}} - 2 \cdot \frac{\ln(\hat{\mathscr{L}}) - \ln(\hat{\mathscr{L}})_{\min}}{\ln(\hat{\mathscr{L}})_{\max} - \ln(\hat{\mathscr{L}})_{\min}}$$

In this step, all the sequences in each dataset are used to calculate the BICs with K varying from 1 to 20. There is usually a reasonable range for the elbow-like parameter selection (Salvador and Chan, 2005b). The final parameter K is chosen based on the minimal normalized BIC. Then $K = 3$ is selected for both lane change and lane keeping in the I-80 dataset; $K = 4$ is selected for lane change and $K = 3$ for lane keeping in the US-101 dataset.

5.4.1. CLASSIFICATION EVALUATION

In order to highlight the effects of surrounding vehicles, the model only considering the information of target vehicles is also studied in the following experiments, which is designated “tgt" for only considering the *target* vehicle. The proposed method is designated “srd" for considering *surrounding* vehicles.

A receiver-operating-characteristic curve is a standard analysis tool to score the performance of a binary classifier system with a varying threshold, i.e. $\mathscr{R}_\mathrm{T}$ here. The area under the curve (AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a negative one (assuming positives rank higher than negatives) (Fawcett, 2006). In this chapter, the AUC means the classification performance of the behavior estimation. The accuracy of behavior estimation is higher when AUC (ranging from 0 to 1) is larger. As shown in Table. 5.3, the AUCs of the “srd" method are higher than the “tgt" method, i.e., the classification results considering surrounding vehicles are more accurate than the results only considering the information of target vehicles.

Table 5.3: Comparison of AUCs.

Cases	I	II	III	IV	V
srd-I-80	**0.9603**	**0.9475**	**0.9418**	**0.9575**	**0.9356**
tgt-I-80	0.9282	0.9064	0.9325	0.9046	0.9182
srd-US-101	**0.9173**	**0.9295**	**0.9270**	**0.9358**	**0.9163**
tgt-US-101	0.9065	0.9167	0.9058	0.8980	0.9007

Table 5.4: Performance index comparison at FPR = 5%.

Dataset		I-80						US-101					
Cases		I	II	III	IV	V	Average	I	II	III	IV	V	Average
TPR	srd	**0.9158**	**0.8055**	0.8056	**0.8425**	**0.8037**	**0.8346**	**0.8091**	0.7478	0.8108	**0.8091**	**0.7727**	**0.7898**
	tgt	0.7757	0.7407	**0.8241**	0.5278	0.6168	0.6971	0.6546	**0.8378**	**0.8738**	0.5909	0.7636	0.7441
FPR	srd	**0.0654**	**0.0648**	0.0740	0.0740	**0.0654**	**0.0688**	**0.0636**	**0.0811**	**0.0811**	**0.0909**	**0.0727**	**0.0778**
	tgt	0.0841	0.0741	**0.0648**	**0.0648**	**0.0654**	0.0706	0.0909	0.0991	0.0901	0.1000	0.0909	0.0942
ACC	srd	**0.9252**	**0.8703**	0.8657	**0.8842**	**0.8691**	**0.8829**	**0.8727**	0.8333	0.8648	**0.8591**	**0.8501**	**0.8561**
	tgt	0.8458	0.8333	**0.8796**	0.7315	0.7757	0.8132	0.7818	**0.8694**	**0.8919**	0.7455	0.8364	0.8249
PRE	srd	**0.9333**	**0.9255**	0.9157	**0.9191**	**0.9247**	**0.9237**	**0.9271**	**0.9022**	**0.9091**	**0.8989**	**0.9139**	**0.9103**
	tgt	0.9022	0.9091	**0.9271**	0.8906	0.9041	0.9066	0.8781	0.8942	0.9066	0.8553	0.8936	0.8855
F_1	srd	**0.9245**	**0.8614**	0.8571	**0.8792**	**0.8600**	**0.8765**	**0.8641**	0.8177	0.8571	**0.8516**	**0.8374**	**0.8456**
	tgt	0.8342	0.8163	**0.8725**	0.6627	0.7333	0.7838	0.7501	**0.8651**	**0.8899**	0.6989	0.8235	0.8055

Besides the AUC evaluation, the following quantitative metrics are also introduced for a comprehensive evaluation:

- True Positive Rate (TPR), also named *Recall*, is the fraction of events classified correctly out of all true events, i.e.,

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$$

where TP means true positive and FN means false negative (missed detection).

- False Positive Rate (FPR) is the fraction of events classified wrongly out of all false events, i.e.,

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$$

where FP means false positive (false alarm) and TN means true negative.

- Accuracy (ACC) is the fraction of correctly classified events out of all testing events. It is defined by

$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$$

- Precision (PRE) is the fraction of events classified correctly out of all events predicted to be positive, i.e.,

$$\mathrm{PRE} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$$

- F1 Score is the harmonic mean of the precision and the recall, i.e.,

$$F_1 = 2 \times \frac{\mathrm{PRE} \times \mathrm{TPR}}{\mathrm{PRE}+\mathrm{TPR}}$$

Note that the thresholds are determined by choosing FPR = 5% in the training data. The thresholds are then used for the final evaluation in the testing set (see the results reported in Table 5.4). The evaluation results show that the proposed method considering the information of surrounding vehicles achieves better performance than the method only considering the target vehicle.

5.4.2. LANE CHANGE PREDICTION

A further challenge is to predict lane change before the target vehicle crosses lane lines. We define the prediction time as

$$\tau_{\mathrm{t}} = t_{\mathrm{e}} - t_{\mathrm{p}}$$

where t_{e} represents the ending time of a scenario and t_{p} is the first instant when a label of lane change is reached. In the testing dataset, t_{e} is the time when the target vehicle crosses the lane lines and t_{p} is the time when the behavior of lane change is estimated. When the ratio $\mathscr{R}$ changes across the threshold, the final driving behavior is estimated as lane change. In addition, if the final output behavior remains lane change until the end of the scenario, the prediction time is obtained as the period between t_{p} and t_{e}.

Table 5.5: Lane change prediction time τ_{t} in seconds.

Cases	I	II	III	IV	V	Average
srd-I-80	5.16	5.21	4.97	3.11	3.49	**4.39**
tgt-I-80	4.12	3.42	2.99	2.58	2.49	3.12
srd-US-101	4.67	4.96	5.38	4.24	4.43	**4.73**
tgt-US-101	2.67	2.81	3.12	2.23	2.41	2.65

Table 5.5 compares the average prediction time between "srd" and "tgt", which demonstrates that the proposed method is able to predict the intention of target vehicles earlier. Moreover, a comparison with lane change prediction using SVM (Kumar et al., 2013) is conducted. The results of the proposed method using GMM-HMM are better because the driving behavior is a time series and previous states are related to current and future states. SVM is a classifier that can only input constant dimensions of variables and is thus unable to model the effects of time series effects. The comparison results are shown in Figure 5.3 and Figure 5.4. The proposed method has an approximately 80% true positive rate of predicting the behavior of lane change 0.5s in advance and retains a 60% true positive rate up to 4s before the lane change occurs. Furthermore, the proposed method also has the lowest false positive rate while the SVM method produces over 20% false positive rate.

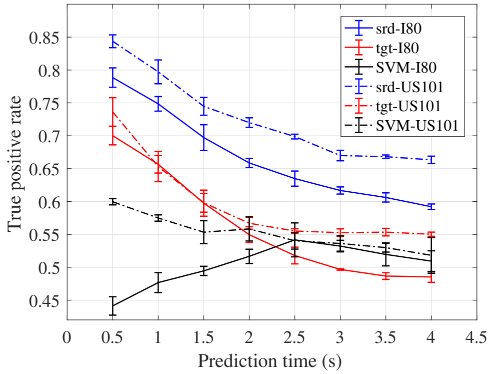


Figure 5.3: Prediction time and true positive rate of lane-change behavior in both dataset.

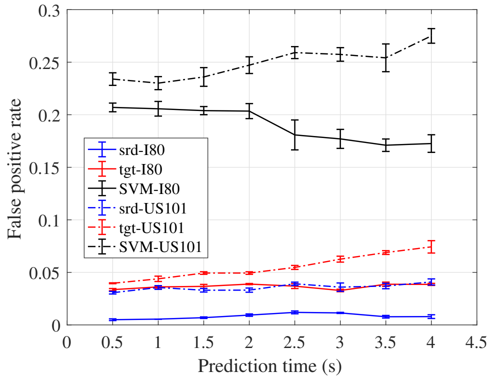
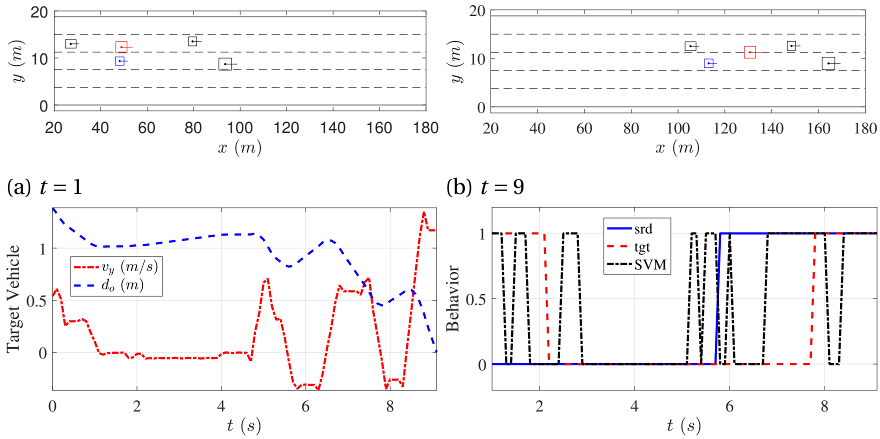


Figure 5.4: Prediction time and false positive rate of lane-change behavior in both dataset.

A detailed example in Figure 5.5 shows that the driving behavior cannot be correctly estimated by only considering the information of the target vehicle. Note that lane keeping is 0 and lane change is 1. In the first second of this scenario, the target vehicle is



(a) $t = 1$

(b) $t = 9$

(c) Lateral speed and lateral offset of the target vehicle

(d) Driving behavior output, 0 means lane keeping, 1 means lane change

Figure 5.5: An example of the proposed behavior estimation method. The red vehicle is the target vehicle. d_0 is the lateral distance of the target vehicle. The time point of a “partial" lane change of the target vehicle at $t = 9s$ is actually considered as the ending of its lane change intention. We assume the true lane change will happen after this moment and this work is trying to do a prediction as early as possible before this moment.

shifting to the right, but it is not lane-change behavior because there is a vehicle in the lane to its right. If only the information of the target vehicle is considered, the algorithm may estimate that the target vehicle is changing lanes even though the target vehicle cannot do so. Therefore, accurate behavior estimation requires considering the traffic situation around the target vehicle. Moreover, due to the lack of modeling of the time-series sequences, the SVM is not stable and cannot make any estimation or prediction without filtering.

5.4.3. CAR-FOLLOWING TESTING RESULTS

The scenarios containing the host and target vehicles in the NGSIM dataset are extracted for the car-following control test. The information about the surrounding vehicles is used as the observation of the host vehicle. The parameters of the MPC are listed in Table 5.6.

As shown in Table 5.7, five metrics are selected to evaluate the proposed method and three methods are compared to demonstrate the influence of the cut-in situations. The proposed method is denoted by “srd-MPC", which means the intention of the target vehicle is estimated by considering the information of all the surrounding vehicles. The method “tgt-MPC" represents the MPC controller with the intention estimated only using the information of the target vehicle. The method “Only-MPC" is the pure MPC method without considering the cut-in intentions of target vehicles. The speeds, accelerations and jerks listed in Table 5.7 are the average value in each test. The hazard index is defined as

$$\mathrm{HI} = \exp\left(-(\Delta x / h_1)^{h_2}\right)$$

Table 5.6: Parameters in MPC.

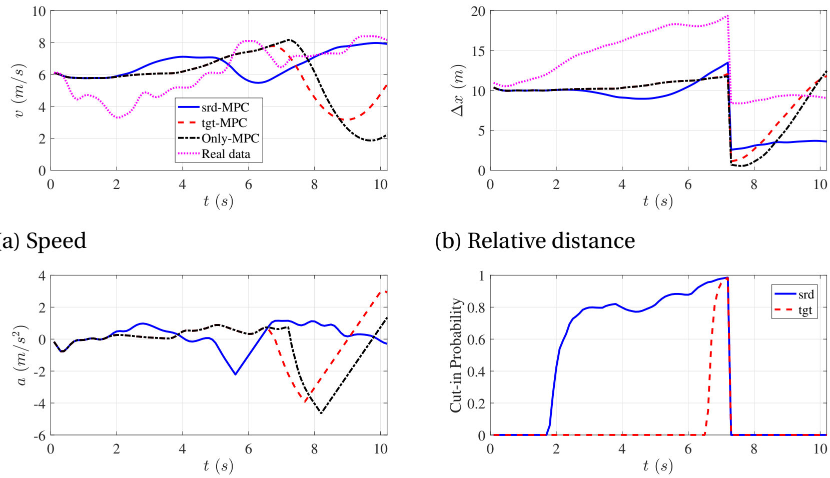
Variables	Values	Units
Δt	0.1	s
N_p	20	–
ω_c	10	–
d_0	6	m
v_max	30	m/s
$\tau_0, \tau_\text{h1}, \tau_\text{h2}$	0.5, 1, 3	s
$\omega_\text{d}, \omega_\text{v}, \omega_\text{a}, \omega_\text{j}$	0.01, 0.02, 0.01, 0.05	–
$a_\text{min}, a_\text{max}$	−4, 6	m/s^2
$j_\text{min}, j_\text{max}$	−0.3, 0.3	m/s^3

Table 5.7: Performance index comparison of MPCs.

Dataset		I-80						US-101					
Cases		I	II	III	IV	V	Average	I	II	III	IV	V	Average
v_h (m/s)	srd-MPC	6.3667	7.5950	6.2163	6.0926	6.1791	6.4899	10.1505	10.4960	10.2020	10.9406	9.7350	10.3048
	tgt-MPC	6.3292	7.5994	6.1411	5.8433	5.9667	6.3759	10.3989	10.7071	10.5165	11.0393	9.8605	10.5045
	Only-MPC	6.9295	7.5845	6.2827	6.0988	6.2823	**6.6356**	10.6237	10.9072	10.6093	11.2079	9.8176	**10.6331**
a_h (m/s^2)	srd-MPC	1.1624	1.0795	1.1589	1.1845	1.2646	**1.1700**	1.1609	1.3325	1.0661	1.7717	1.4109	**1.3484**
	tgt-MPC	1.1974	1.5522	1.3786	1.2096	1.4739	1.3623	1.1632	1.6061	1.4135	1.8874	1.4795	1.5099
	Only-MPC	1.4067	1.5785	1.4746	1.3482	1.4555	1.4527	1.4798	1.6183	1.4058	1.9159	1.7556	1.6351
Δa_h (m/s^3)	srd-MPC	0.1253	0.1399	0.1378	0.1409	0.1548	**0.1397**	0.1245	0.1526	0.1145	0.1787	0.1550	**0.1451**
	tgt-MPC	0.1263	0.1836	0.1569	0.1498	0.1783	0.1590	0.1320	0.1710	0.1511	0.1841	0.1619	0.1600
	Only-MPC	0.1625	0.1892	0.1732	0.1606	0.1734	0.1718	0.1698	0.1730	0.1515	0.1827	0.1850	0.1724
HI	srd-MPC	0.0310	0.0214	0.2641	0.3888	0.4185	**0.2248**	0.2664	0.3675	0.1517	1.0301	0.4928	**0.4617**
	tgt-MPC	0.1245	0.4692	0.2650	0.4297	0.8305	0.4238	0.2865	0.8062	0.7836	1.2306	0.7578	0.7729
	Only-MPC	0.6393	0.4705	0.2821	0.6097	0.9959	0.5995	0.6062	0.8269	1.1758	1.3458	1.0394	0.9988
CR	srd-MPC	0/29	0/22	1/25	2/25	2/21	**0.0430**	2/35	2/28	1/40	6/31	2/36	**0.0805**
	tgt-MPC	1/29	3/22	1/25	3/25	3/21	0.0947	2/35	6/28	5/40	8/31	4/36	0.1531
	Only-MPC	4/29	3/22	1/25	4/25	4/21	0.1330	4/35	6/28	8/40	8/31	6/36	0.1907

which represents the degree of a rear end collision (Dou et al., 2016). The values of h_1 and h_2 are fitted by the highway naturalistic driving data in (Liu et al., 2017a). The collision rate (CR) represents the collision numbers in the simulation of the host vehicle. The results show that the average speed of the proposed method is close to the traditional MPC. With the intention estimation of the target vehicle, the effect of a sudden change of the leading vehicle is smoothed. Meanwhile, the hazard index and the collision rate of the proposed method are much lower than those of the other methods. Note that the trajectories of cut-in vehicles are used as real stochastic inputs, though fixed in the dataset, to experimentally demonstrate the collision avoidance control of the proposed method. The on-line interaction between host vehicles and cut-in vehicles are omitted as a fundamental assumption.

Two detailed examples from the testing data are illustrated to explain the advantage of the proposed method in Figure 5.6 and Figure 5.7, where the real data are from human drivers in the dataset. The first example is a cut-in scenario in the I-80 dataset as shown in Figure 5.6. In this scenario, the cut-in behavior happens when the target vehicle is slow and wants to give way to a faster following vehicle. As shown in Figure 5.7d, the lane change intention of the target vehicle is detected at 1.8 s by the proposed method, and the target vehicle crosses the lane lines at 7.3 s, where the sudden change of relative distance is shown in Figure 5.7b. Such an intention is detected at 6.6 s using the target vehicle information only. By using the proposed method, the host vehicle is able to take an earlier intervention control of slowing down before the cut-in, and therefore obtains smooth accelerations and avoids a hard brake.



(a) Speed (b) Relative distance

(c) Acceleration (d) Cut-in Probability

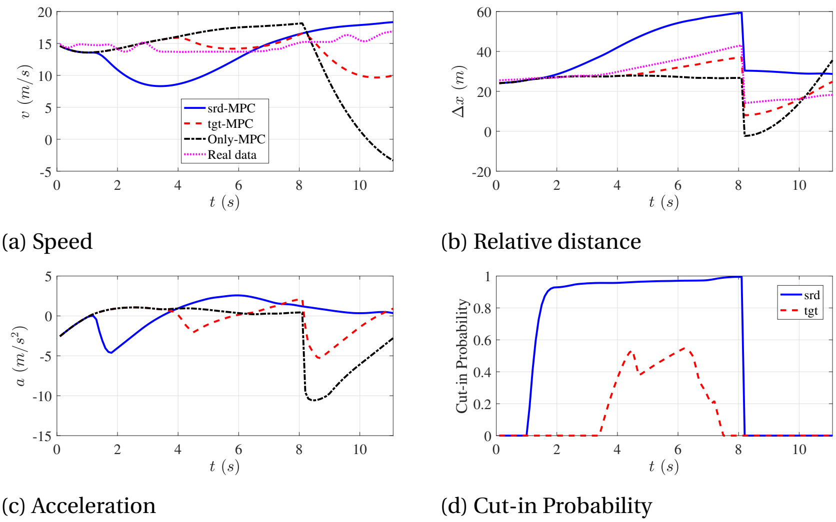
Figure 5.6: An example of the car-following simulation in the I-80 dataset.

Another example from the US-101 dataset is shown in Figure 5.7. The target vehicle in this scenario is trying to merge into the lane of the host vehicle to speed up. The proposed method estimates the cut-in behavior at 1.1 s, while the target vehicle crosses the lane lines at 8.2 s. Similarly to the last scenario, an earlier and smoother control can be seen in the Jerk subplot. Without the intention estimation, the host vehicle controlled by the pure MPC fails to avoid the collision due to the sudden cut-in.

5.5. CONCLUSIONS

This chapter develops a car-following control method with the estimation of the lane-change behavior of other traffic participants. Multivariate time series data from the target vehicle and its surrounding vehicles are used to build two continuous HMMs representing the behavior of lane change and lane keeping. A threshold-based classification method is used to estimate the target vehicle's behavior. In the meantime, a cut-in probability is calculated based on the behavior estimation and the MPC method is then applied to optimize the car-following behavior of the host vehicle. The behavior model of the target vehicle is able to achieve over 85% of the true positive rate and the lane change behavior is predicted about 4 seconds before the target vehicle crosses the lane lines. The proposed intention-based MPC achieves superior performance of safety and ride comfort.

In future, we will investigate the strategies based on intention prediction in more complicated scenarios like at intersections. The interpretation of the complicated model is also a research line. An insightful model such as timed automaton would act as a



(a) Speed (b) Relative distance

(c) Acceleration (d) Cut-in Probability

Figure 5.7: An example of the car-following simulation in the US-101 dataset.

promising alternative solution.

6

Learning automaton for diagnosing a control system

In the last chapter, we discuss the application of combining stochastic environment learning and model predictive control to enhance the safety of autonomous driving cars.

In this chapter, we continue the discussion about the safety problem of intelligent control systems. The context is under the intrusion detection for safety-critical cyber-physical systems. We propose a novel model called TABOR combining timed automata with a Bayesian network. Timed automata are used for modeling the regularity of signals, while the Bayesian network is used for modeling the causality between multiple sensors and actuators. TABOR is highly explainable and capable of localizing faulty components.

The material in this chapter has appeared in
Qin Lin, Sridha Adepu, Sicco Verwer, and Aditya Mathur. Tabor: Agraphical model-based approach for anomaly detection in industrial control systems. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pages 525–536. ACM, 2018

6.1. Introduction

The protection of industrial control systems (ICS) (Stouffer et al., 2011; icsCERTAdvisory) for public infrastructure such as power, water treatment, and transportation systems is of utmost importance due to the significant damage a potential attack may cause. Often these systems are vulnerable to attacks due to the presence of cyber components such as Supervisory Control and Data Acquisition (SCADA) workstations, Human Machine Interface (HMI), Programmable Logic Controllers (PLCs) and the underlying communications network. Attacks are a result of exploitation of one or more vulnerabilities (Wilhoit and Hara, 2015) in an ICS. Such vulnerabilities might be due to lack of access control in the system (Adepu et al., 2017). Software vulnerabilities could be in the PLCs, SCADA software systems, and weaknesses in the communication channels. The compromise or destruction of an ICS would impact society in far- reaching ways. For instance, a blackout (Lipovsky, 2016) caused by an attack targeted at a power system ICS would cause monetary losses to all the people served and businesses. Moreover, such an attack could cause cascading failures (Koç et al., 2014), harming large communities such as entire cities. Attacks on ICS can have a significant impact depending on the type of attack and its location. The increase in successful cyber attacks on ICS (Cobb, 2015; Lipovsky, 2016; Weinberger, 2011), and many unsuccessful attempts (ics-cert), points to the importance of research in the security of ICS with the goal of making it resilient to cyber attacks. Attacks on ICS are increasing each year and perhaps leading towards cyber warfare with critical infrastructure as key targets. In this chapter, we aim at detecting intrusions by only observing the physical process under the control of an ICS.

Existing approaches dealing with cyber attack detection in cyber physical systems (CPS)[1] include signature-based detection (Oman and Phillips, 2007; Gao and Morris, 2014), verification (Zheng and Julien, 2015; Clarke and Zuliani, 2011), behavior specification (Adepu and Mathur, 2016b), and machine learning (Junejo and Goh, 2016; Goh et al., 2017). Signature-based methods require an up-to-date signature dictionary of all known attacks, which is becoming increasingly infeasible due to the growing number of unknown threats. Verification methods basically use a formal model to test on a source code level whether certain signals show large deviations from the values specified in the system's design. Although powerful, full verification based on the source code is often infeasible due to the state-explosion problem, in which the resulting model becomes too large to analyze. Behavior specification-based methods require a precise understanding of how the CPS behaves. Such knowledge can be expensive to obtain. Once obtained, however, it can detect many attacks because it uses detailed models of the underlying physical processes. This approach is sensitive to noise caused by dynamic operating environments, aging or other evolvements of the facility in question, and inaccuracies/incompleteness of source documents such as operation manuals (Junejo and Goh, 2016). Most existing machine learning approaches focus on detecting anomalies in feature space, i.e., looking at data points with large deviation from normal space. These require little system knowledge and can detect a large range of attacks. A significant shortcoming of the currently applied machine learning methods is that they provide little insight into the system and no explanation of detection results.

[1] ICS,CPS used interchangeably in all over the chapter

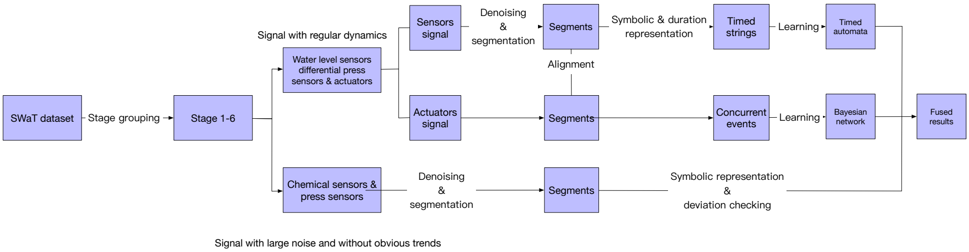


Figure 6.1: Flowchart of TABOR. The sensors and actuators in SWaT are divided into sub-models due to different stages and functionalities. The behaviors of water level and differential press signal show more regular patterns in the dataset, thus they are learned using timed automata. The smallest unit for anomaly detection is one segment, which is considered as an event. The general system alarms an instruction when any sub-model detects anomalies.

In this chapter, we attempt to respond to two key challenges in applying machine learning techniques in the context of a CPS. First: Can we explain the outcome of attack detection, i.e., *why* is this an anomaly? Second: Can we *localize* the anomaly, i.e., which sensors and actuators are potentially under attack? These two questions are of importance for operators who need to diagnose the abnormal behavior and to undertake one of possibly many follow-up safety actions. To deal with the aforementioned problems, an insightful graphical model (Timed Automata and Bayesian netwORk–TABOR) is learned from the normal operational observation of an ICS. The method used in TABOR is illustrated in Figure 6.1 using a flowchart.

Subprocesses of the entire ICS are modeled. Sets of sensors and actuators in the ICS are partitioned into groups based on their functionalities in order to deal with high dimension and complexity of the problem. Signals from the sensors are symbolically represented and learned using timed automata (TA) to discover the underlying dynamical fluctuating behavior of the water level and other sensors. The states in the TA are associated with other actuator's states by dependency/causality inference using the Bayesian network. Irregular patterns and dependencies that do not adhere to the learned model from normal behavior, are considered anomalies. The contributions of this chapter are listed as follows:

1. The proposed model provides a solution for the interpretation and localization of anomalies. The detected anomalous patterns can be located precisely to process, sensors, or actuators. The model is visualizable and interpretable, thus enabling a better understanding of the system and verification of the model itself.

2. More attack scenarios are successfully detected compared to those detected using methods based on deep neural network (DNN) and the support vector machine (SVM) available in the literature.

3. To the best of our knowledge, this is the first work to combine timed automata learning and Bayesian network inference for anomaly detection in a CPS. Techniques used here are not limited to a water plant but also applicable to other CPSs

The remainder of this chapter is organized as follows. Related work is discussed in Section 6.2. The dataset and the attack scenarios are briefly explained in Section 6.3. The proposed method is discussed in Sections 6.4 and 6.5. Analysis of data from the experiments is in Section 6.6. Concluding remarks and future work are in Section 6.7.

6.2. RELATED WORK

The study reported here focuses on cyber attacks on CPS that result in deliberate sensor and actuator data manipulation. There exist several techniques for detecting process anomalies in CPS. These anomalies might happen due to sensor and actuator manipulation in communication channels. Researchers have presented challenges in safety and security against cyber attacks that need to be addressed while designing a CPS. In (Cardenas et al., 2008) the authors have presented evolution of Industrial Control Systems (ICS) to emerging CPS with the use of ICT technologies, and potential vulnerabilities and design challenges. Lee (Lee, 2008) presents problems in computing and network technologies for full-fledged design of emerging CPS.

CPAC (Etigowni et al., 2016) presents a stateful detection mechanism to detect attacks against control systems. The Weaselboard (Mulder et al., 2013) uses a PLC backplane to get the sensor data and and actuator commands, and analyses them to prevent zero day vulnerabilities. WeaselBoard (Mulder et al., 2013) has a dedicated device, and detects changes in control settings, sensor values, configuration information, firmware, logic, etc. In (Stankovic, 2016), it is shown how safety-critical systems are interconnected and their complexity. Model-based attack detection schemes in water distribution systems is presented in (Ahmed et al., 2017). It uses a Matlab identification tool to get a model from the data generated in a water distribution system. The data-driven model is helpful in detecting process anomalies. Cardenas et al. (Urbina et al., 2016) have experimented with the use of CUSUM in detecting stealthy attacks. The research on attack detection in ICS is increasing and monitoring the physics of the ICS to detect attacks is also a growing research area. A water control system is modeled using an autoregressive model and a detector (Hadžiosmanović et al., 2014) which track the process variables. Liu et al. presents false data attacks in a power grid (Liu et al., 2009, 2011), state estimation and intelligent attacks against a state estimation. Response and detection are investigated on attacks against chemical plants (Cárdenas et al., 2011).

The RNN is one of the machine learning approaches used for anomaly detection in the SWaT system (Goh et al., 2017). However, due to the expensive training time (one week), they only consider the first out of the six stages of the system. In addition, only 10 attack scenarios are used for the evaluation. Recently, as a follow-up work, the DNN and the one-class SVM models have been applied for anomaly detection in the SWaT system (Inoue et al., 2017). All stages and attack scenarios are considered in this work. Due to the comprehensiveness of this work and the similarity between RNN and DNN, this work is used for comparison with the proposed model.

Formal methods are also powerful tools for specification mining in the CPS. They usually cover *signal level* and *code level* verifications. The code level verification is not related to the research problem in this work. while the signal level verification aims at discovering signal rules, e.g., Signal Temporal Logic (STL) formulas (Jones et al., 2014) from the normal behaviors of the CPS. However, due to the high complexity of the approach, only some simulation cases are considered in their work, which is thus not suitable to deal with the high-dimensional data in the SWaT system. Another main difference lies in the fact that the proposed model is actually a passive grammar learning approach and treats signals using a symbolic representation in the pre-processing step.

Both of the proposed grammar-based and rule-based methods are possible candidates to enrich the invariants (Adepu and Mathur, 2016b) in the CPS. They both are essentially "specification mining" techniques offering an interesting research line of combining statistical machine learning and verification.

Timed automata have been used in the discrete event system domain for model-based diagnosis (Tripakis, 2002; Bouyer et al., 2005). However, it is usually assumed that the plant and the specification model are already obtained. Our work combines model identification and anomaly detection in an integrated framework. In (Maier et al., 2011; Vodenčarević et al., 2011; Niggemann et al., 2012), a (hybrid) timed automaton is learned and used for anomaly detection in production systems. The authors use a bottom-up strategy for timed automaton learning, which is one key difference with our approach. Moreover, signals from sensor and actuator are mixed up and represented as events in their approach, which leads to a states blow-up problem and difficulty in localizing the abnormal sensor/actuator. We have shown the possibility of learning timed automata for anomaly detection in a Digital Video Broadcasting System (Liu et al., 2017b). In the proposed work, a Bayesian network is additionally learned to discover the dependencies between the sensors and the actuators in SWaT.

6.3. INTRODUCTION TO SWAT AND THE DATASET

SWaT is a scaled-down water treatment plant with a small footprint that produces 5 gallons/minute of doubly filtered water. This testbed replicates large modern plants for water treatment such as those found in cities. SWaT has six sub-processes, referred to as *stages*, controlled by six PLCs, as shown in Figure 6.2 (Adepu and Mathur, 2016c).

The architecture of SWaT is well introduced in the literature (Adepu and Mathur, 2016c). Here we recapitulate the functions of the six sub-processes. Stage P1 controls the inflow of raw water to be treated by opening or closing a motorized valve. The raw water tank is treated in the chemical dosing station (stage P2), then flows to another UF (Ultra Filtration) feed water tank in stage P3. A UF feed pump in P3 sends water via UF unit to the RO (Reverse Osmosis) feed water tank in stage P4. Here an RO feed pump sends water through an ultraviolet dechlorination unit controlled by a PLC in stage P4. This step is necessary to remove any free chlorine from the water prior to passing it through the reverse osmosis unit in stage P5. Sodium bi-sulphate (NaHSO3) can be added in stage P4 to control the ORP (Oxidation Reduction Potential). In stage P5, the dechlorinated water is passed through a 2-stage RO filtration unit. The filtered water from the RO unit is stored in the permeate tank and the reject in the UF backwash tank. Stage P6 controls the cleaning of the membranes in the UF unit by turning on or off the UF backwash pump. The backwash cycle is initiated automatically once every 30 minutes and takes less than a minute to complete. A backwash cycle is also initiated if the pressure drop exceeds 0.4 bar, which indicates that the membranes in the UF unit are clogged and need to be cleaned. A differential pressure sensor at stage P3 is used by PLC-3 to obtain the pressure drop across the UF unit.

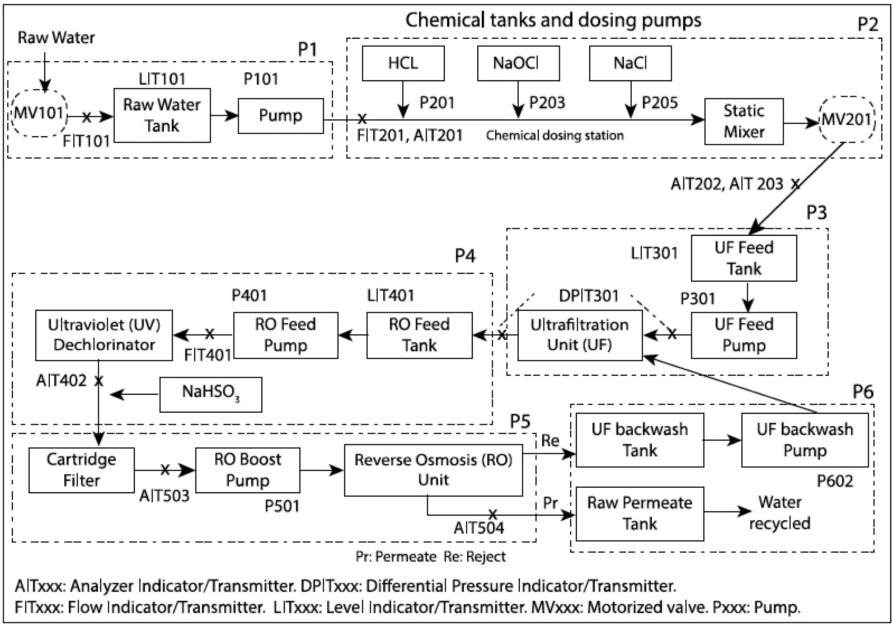
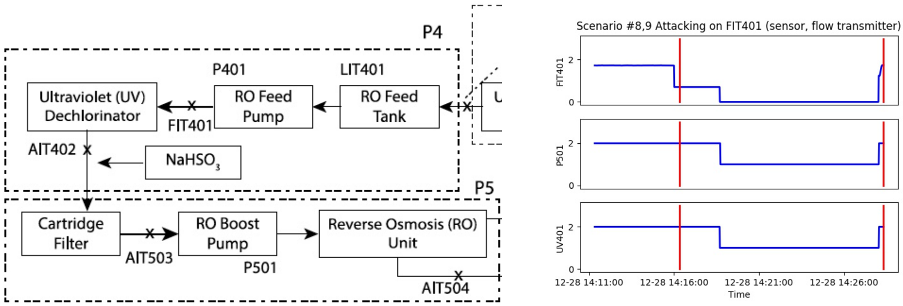


Figure 6.2: SWaT system diagram. The functionality of each stage is as follows: P1: Raw water supply and storage. P2: Pre-treatment via chemical dosing. P3: Ultrafiltration (UF) and backwash. P4: De-Chlorination system. P5: Reverse osmosis (RO). P6: RO permeate transfer, UF backwash and cleaning.

The SWaT dataset (J. et al., 2016) was collected over 11 days of continuous operation. The first 7 days of data were collected under normal operation (without any attacks) while the remaining 4 days of data were collected with 36 designed attack scenarios. All network traffic, physical (sensor and actuator) data were collected. In this chapter, we focus on the detection of the physical process; network traffic data are ignored. The duration of physical recording is from 22/12/2015 4:00:00 PM to 2/1/2016 2:59:59 PM. The dataset contains a total of 53 columns: 1 for *timestamp*, 1 for *label* ('Attack' and 'Normal'), and the remaining 51 for numeric recordings of 51 sensors and actuators. Note that physical data are equally sampled every second. The description of all 36 attack scenarios can be found on the website.[2]



(a) (b)

Figure 6.3: An example of sensor attack on SWaT. Water from the RO tank is sent via an ultraviolet (UV) and a cartridge filter to the next stage (P5). Flow meter FIT401 indicates the flow rate of water from the RO tank to through the UV. An attack (the starting and ending time are indicated via timestamp of two red bars in Figure 6.3b) manipulates the real value (around 2 m^3/h) to 0.7 m^3/h then 0 m^3/h. Actuator UV and Pump 501 are turned off to lead the PLC into believing that there is no water transmitted from the RO feed tank. This subsequently leads to overflow in the RO tank.

6.3.1. ATTACK SCENARIOS

Attacks in the attack dataset were generated based on scenarios reported earlier (Adepu and Mathur, 2016b,c). The attack model is a generalized model (Adepu and Mathur, 2016a) for cyber physical systems with an intent space of an attacker. The attack duration depends on the kind of attack and attacker intent. The duration of each attack varies from 101 seconds to 10 hours. Some attacks are consecutively within a 10-minute gap, while others are performed by leaving time for the system to stabilize. 36 attacks were launched during the data collection process. Based on attack points in each stage, the attacks are divided into: 26 Single Stage Single Point (SSSP) attacks performed on exactly one point in a CPS; 4 Single Stage Multi Point (SSMP) attacks on two or more points but on only one stage; 2 Multi Stage Single Point (MSSP) attacks and 4 Multi Stage Multi Point (MSMP) attacks performed on two or more stages.

All attacks are performed by injecting the process variable values into a Programmable Logic Controller (PLC) leading each PLC to believe that the sensor informa-

[2]https://itrust.sutd.edu.sg/dataset/

tion received is genuine and is not a spoofed value. Some attacks are stealthy (J. et al., 2016) where an attacker changes the sensor values slowly with respect to the process behaviour; other attacks are random in which sensor values shift randomly. A detailed description of the threat model is in (Kang et al., 2016). Figure 6.3 shows an example of an attack in the De-Chlorination stage (P4).

6.4. SIGNAL PROCESSING

This section discusses the pre-processing procedure dealing with the high-dimension and noisy signal in the SWaT system. In Table 6.1, groups of sensors and actuators are split locally in six stages of the system. For sensors labeled LITxxx that measure water levels and those labeled DPIT that measure differential pressure, the sequential behavior, as well as their dependencies on the actuators in the same stage, are learned because of the relatively obvious regular patterns. The signals from the sensors AIT and PIT with large noise and subtle trends are checked only using a model-free approach, i.e., by examining their values and the thresholds. The data imply that the differencing effect of DPIT makes the time series of the PIT signal stationary. Note that several sensors and actuators in P6 are not used in the work reported here because they are not completely used for data collection in SWaT yet. Only the first five stages are considered in this work. In addition the dataset used does not contain any attacks on stage 6.

Table 6.1: Sub-model Split. FIT is simply treated as actuator with two states: closed and open.

Model Number	Stage Number	Sensor	Actuator
1	1	LIT101	MV101, FIT101, P101, P102
2	2	AIT201	FIT201, MV201, P201
3	2	AIT202	FIT201, MV201, P201, P203, P205
4	2	AIT203	FIT201, MV201, P201, P203, P205
5	3	DPIT301	FIT301, MV301, MV302, MV303, MV304, P302
6	3	LIT301	FIT301, MV301, MV302, MV303, MV304, P302
7	4	AIT401	FIT401,P-402,P-403,UV-401
8	4	AIT402	FIT401,P-402,P-403,UV-401
9	4	LIT401	FIT401,P-402,P-403,UV-401
10	5	AIT501	FIT501,FIT502,FIT503,FIT504,P501
11	5	AIT502	FIT501,FIT502,FIT503,FIT504,P501
12	5	AIT503	FIT501,FIT502,FIT503,FIT504,P501
13	5	AIT504	FIT501,FIT502,FIT503,FIT504,P501
14	5	PIT501	FIT501,FIT502,FIT503,FIT504,P501
15	5	PIT502	FIT501,FIT502,FIT503,FIT504,P501
16	5	PIT503	FIT501,FIT502,FIT503,FIT504,P501

6.4.1. DENOISING

The sensor signal has already been denoised by a hard filter in each stage. However, spikes are still observed and thus pose a challenge to the following learning procedure. The one-dimensional time series of a sensor signal is defined as:

$$\mathbf{x}[n] = [x_1, x_2, \cdots, x_n] \tag{6.1}$$

A naive averaging filter is applied here for denoising. The denoised time series is defined as

$$\bar{\mathbf{x}}[w] = [\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_w] \tag{6.2}$$

The ith element of $\bar{\mathbf{x}}$ is calculated by:

$$\bar{x}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} x_j \tag{6.3}$$

For simplicity and clarity, it is assumed that n is divisible by w. If not, it can be simply modified by appending an additional chunk to $\bar{\mathbf{x}}$ averaging the remainders in $\mathbf{x}$. The original and the denoised signal are shown in Figure 6.4.

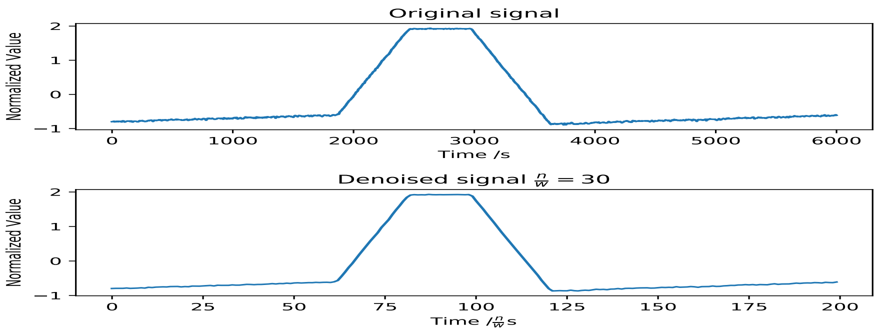


Figure 6.4: Denoising by an averaging processing.

6.4.2. SEGMENTATION

Representation is key to efficient and effective solutions to time series data mining. As one of the most commonly used preprocessing methods, piecewise-linear representation (PLR) has been used by various researchers to support clustering, classification, indexing, and association rule mining of time series data (Keogh et al., 2001). In this chapter, a Sliding WIndow based on Differential sEgmentation (SWIDE) algorithm is used for the piecewise-linear approximation of the sensor signal. Pseudo code of SWIDE is shown in Algorithm 7. A segmented sensor signal is shown in Figure 6.5.

6.4.3. ALIGNMENT

A quantile clustering algorithm is used for the discretization and the symbolic representation of the sensor signal, i.e., letting each bin have equal frequency. The inputs

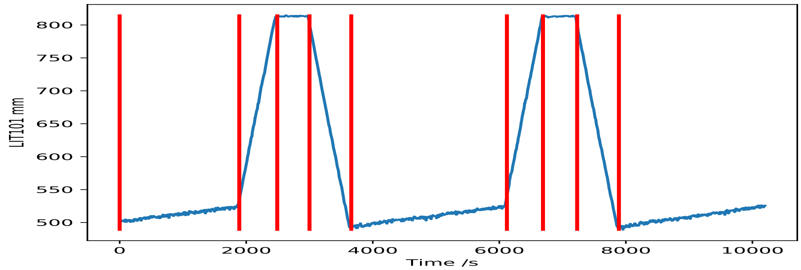


Figure 6.5: Segmentation: The original noisy signal is shown; the denoised signal is used as input to SWIDE for obtaining more robust segmentation results.

Algorithm 7 SWIDE algorithm:

Require: Denoised time series data $\bar{\mathbf{x}}$, max_error ϵ
Ensure: PLR with K segments Seg
  $anchor = 1$
  $diff\_seg\_mean = 0$
  while not finished segmenting time series **do**
    $i = 2$
    while $abs(\bar{x}_i - \bar{x}_{i-1})$−diff_seg_mean) $\leq \epsilon$ **do**
      $i = i + 1$
      $diff\_seg\_mean = update\_diff\_mean(\bar{x}[anchor : anchor + (i-1)])$ ▷ recalculate the mean differential value
    end while
    $Seg = concat(Seg, create\_segment(\bar{x}[anchor : anchor + (i-1)])$ ▷ add this segment
    $anchor = anchor + i$
  end while

---

to the clustering algorithm are the differential values of the segments $diff\_seg\_mean$ obtained in the segmentation step. The subplot of LIT101 in Figure 6.6 shows the discretized signal represented as four letters. The number of clusters is set by looking at the trends in the training data. The trends of slow up (SU), quick up (QU), staying constant (SC), and quick down (QD) are obviously visible and interpreted. They are interpreted by human beings once the number of clusters and the average value in each cluster are determined. Such a representation in natural language is straightforward to boost the interpretation of the model. Meanwhile the corresponding status of the actuators is obtained using the timestamps from the sensor signal's segments. Figure 6.6 shows the alignment of the sensors and the actuators in P1. The durations of events are implicitly represented along with events into timed strings, which are fed to the timed automata learning algorithm. The concurrent events from the aligned sensor and actuator values are input to Bayesian network learning.

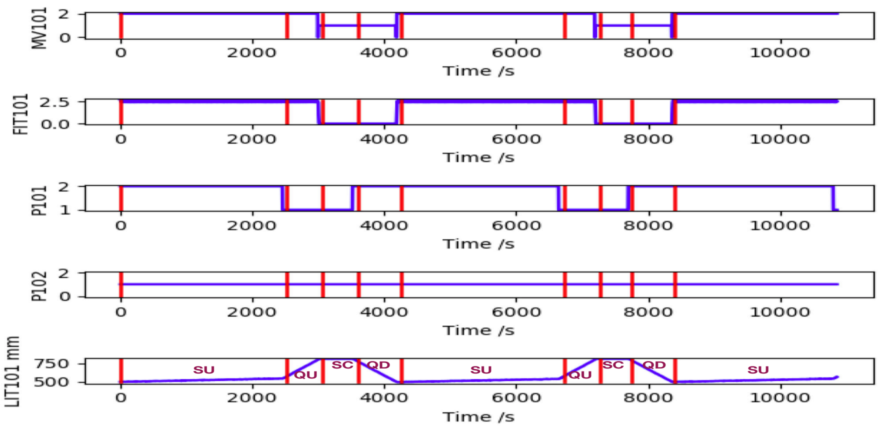


Figure 6.6: Alignment of the sensors and the actuators in P1 based on segmentation and clustering results. The timed string of the sensor is: (SU,2520) (QU,540) (SC,540) (QD,660) (SU,2460) (QU,540) (SC,480) (QD,660) (SU,2460). The possible misplacement of segments is due to the noise in the original signal.

6.5. TABOR LEARNING

Timed automaton, Bayesian network, and their learning algorithms are explained in this section. The input to the timed automata learning procedure consists of sentences of timed strings. One sentence consists of two full cycles to better capture any looping behavior in the state machine. Two consecutive sentences have one cycle as overlapping. The input to the Bayesian network learning procedure consists of just the data points of concurrent events from the alignment of the sensors and actuators.

6

6.5.1. PROBABILISTIC DETERMINISTIC REAL TIMED AUTOMATON

Here we introduce the probabilistic deterministic finite automaton (PDFA), which is more commonly used in practice, and then move to the probabilistic deterministic real timed automaton (PDRTA), which is the model used in this work. The PDFA defined in Definition 3 is a generic model for discrete events (similar to a Hidden Markov Model).

Definition 3. A PDFA is a 5-tuple $\langle Q, \Sigma, \delta, \pi, q_0 \rangle$, where Q is a finite set of states, Σ is a finite alphabet of observable symbols (events), $\delta : Q \times \Sigma \rightarrow Q$ is the transition function from a state-symbol pair to the next state, $\pi : Q \times \Sigma \rightarrow [0,1]$ is the probability of the emitted symbol given a state, and q_0 is the initial state.

Sequences of symbols translate to paths over states starting from the initial state q_0. The probability of such a sequence is obtained by multiplying all the state-symbol probabilities along such a path. Time information is also relevant in many real-world applications of automata. The timing of actions, or lifetime, is important for characterizing behaviors, and is also considered as a feature for anomaly analysis in this chapter. An algorithm for efficient learning of timed automata is proposed in Ref. (Verwer et al., 2006, 2010a). This algorithm uses an *explicit* representation of such time constraints. Discrete events are represented by timed strings $(a_1, t_1)(a_2, t_2)\cdots(a_n, t_n)$, where a_i is a discrete event occurring with t_i time delay since the $(i-1)$th event. In this chapter, t_i is the duration of each event a_i. A PDRTA is formally defined as follows.

Definition 4. A PDRTA is a 4-tuple $\langle\mathscr{A},\mathscr{E},\mathscr{T},\mathscr{H}\rangle$, where $\mathscr{A} = \langle Q,\Sigma,\Delta,q_0\rangle$ is a 4-tuple defining the machine structure: Q is a finite set of states, Σ is a finite alphabet, Δ is a finite set of transitions, and $q_0 \in Q$ is the initial state. $\mathscr{E}$ and $\mathscr{T}$ are the event and time probability distributions, respectively. $\mathscr{E}: Q\times\Sigma \to [0,1]$ returns the probability of generating/observing a given event in a given state. $\mathscr{T}: Q\times\mathscr{H} \to [0,1]$ returns the same but for a given time range $[m,m'] \in \mathscr{H}$, where $\mathscr{H}$ is a finite set of non-overlapping intervals in $\mathbb{R}_+$. A transition $\delta \in \Delta$ in a PDRTA is a tuple $\langle q,q',a,[m,m']\rangle$, where $q,q' \in Q$ are the source and target states, $a \in \Sigma$ is a symbol and $[m,m']$ is a temporal guard.

In a PDFA and a PDRTA, the states are *latent variables* that cannot be directly observed in strings, but have to be estimated by using a learning method. The state transition in a PDFA is triggered only by an event. However, in a PDRTA, it is triggered when both an event and its timing are validated (inside a time range/guard). Therefore, a PDRTA is essentially a timed variant of a PDFA.

6.5.2. LEARNING PDRTA

A state-of-the-art machine learning algorithm named RTI+ is used to learn human behaviors from unlabeled data (Verwer, 2010a; Lin et al., 2018b). The traditional state machine learning algorithm starts by building a large tree-shaped automaton called an augmented prefix tree acceptor (APTA) from a sample of input strings. Every state of this tree can be reached by exactly one untimed string and therefore encodes exactly the input sample. For timed automaton learning, the initial values of the lower and upper bounds of all time guards are set to be the minimum t_{min} and maximum t_{max} time values from the input samples S. Figure 6.7 illustrates a timed APTA (TAPTA) from timed strings (a modified example from Ref. (Verwer, 2010a)).

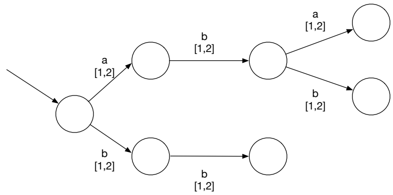


Figure 6.7: TAPTA constructed from the timed input sample: $S = (a,1),(a,1)(b,2)(b,1),(b,2)(b,1),(a,1)(b,1)(a,1),(b,2),(b,1)(b,1)$. It basically continually adds nodes for new symbols in each node.

State merges and transition splits are two main operations of structure and temporal guards learning in RTI+. A split of a transition (see an example in Figure 6.8) $\delta = \langle q,q',a,[m,m']\rangle$ at time t creates two new transitions $\langle q,q_1,a,[m,t]\rangle$ and $\langle q,q_2,a,[t+1,m']\rangle$. The target states q_1 and q_2 are the roots of two new prefix trees that are reconstructed based on the input sample.

The algorithm also greedily merges pairs of states (q,q') in this tree, forming an increasingly smaller machine that generalizes over samples, as shown in Figure 6.9. Be-

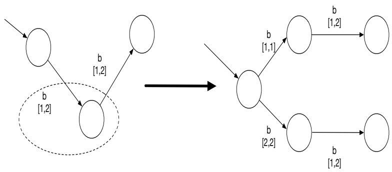


Figure 6.8: A split of a part of the TAPTA from Figure 6.7.

cause PDRTAs are deterministic, for every event $e \in \Sigma$ the states that are reached from q and q' have to be merged as well– also known as the determinization process.

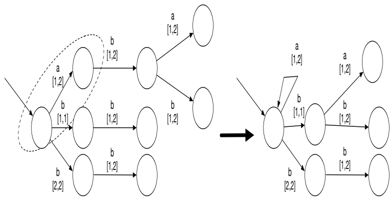


Figure 6.9: A merge operation of TAPTA after the split from Figure 6.8

Note that these examples are only a possible split and merge illustrating how to conduct these operations. The algorithm uses a likelihood-ratio statistical test to decide whether to split/merge or not (Verwer et al., 2010a). A hypothesis H is called nested within another hypothesis H' if the possible distributions under H form a strict subset of the possible distributions under H'. By definition, H' has more unconstrained parameters (or degrees of freedom) than H ($r' > r$). In our case, H is the model after merge (resp. before a split) and H' is the model before a merge (resp. after a split). Given two hypotheses H and H' such that H is nested in H', and a data set S, the likelihood ratio test statistic is computed by:

$$LR = \frac{LK(S,H)}{LK(S,H')} \tag{6.4}$$

where the likelihood LK estimates how likely it is that S is generated by the corresponding hypothesis. The random variable $y = -2\ln(LR)$ is asymptotically $\chi^2(r'-r)$ distributed (Wilks, 1938). The p-value is the probability that a random value in $\chi^2(r'-r)$ would be greater than or equal to the observed value y by chance. If it is smaller than 0.05, H and H' are significantly different with 95% confidence so that a split operation is accepted. In addition, a merge is accepted whenever the model after the merge is not significantly different from the model before the merge, since they are supposed to have similar or compatible stochastic and timed behaviors. Note that the current version of RTI+ tries to model time and events distributions independently. An overview of RTI+ is in Algorithm 8. The model learned of the LIT101 sensor signal is shown in Figure 6.10. Any testing sequence that is not fired by the learned TA is alarmed as an anomaly, i.e., the abnormal event lasts until the end of the sequence. There are two typical types of alarms in TA: "event error" (symbol that can not be fired for transition in the given state) and "timing error" (symbol's timing is outside the valid time guard).

6

Algorithm 8 Data identification with RTI+:

Require: A (multi-)set of timed strings S_+
Ensure: A small PDRTA $\mathscr{A}$ for S_+
  Construct a timed prefix $\mathscr{A}$ tree from S_+, let $Q' = \emptyset$
  for all all transitions $\delta = \langle q, q', a, [m, m'] \rangle$ from $\mathscr{A}$, **do**
    Evaluate all possible merges of q' with states from Q'
    Evaluate all possible splits of δ
    if the lowest split p-value< 0.05 **then**
      perform this split
    else if the highest merge p-value> 0.05 **then**
      perform this merge
    else
      add q to Q'
    end if
  end for

6.5.3. LEARNING BAYESIAN NETWORK

A Bayesian network (BN) is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). In this chapter, the BN is learned to model the dependencies among random variables from the sensors and the actuators in the local process; an example is illustrated in Figure 6.11, which is the BN learned from P1. A BN consists of the graph structure (representing the dependencies) and the parameters. The parameters are represented by conditional probability distribution, summarizing the probability distribution of a node given its parents. In this chapter, variables from sensor and actuator are discretized, thus the probability distribution of each node is actually a conditional probability distribution (CPD) table, also see the CPD in Figure 6.11.

Bayesian network learning includes structure learning and parameter learning. In this chapter, a greedy search algorithm K2 (Cooper and Herskovits, 1992) is used for the structure learning. The general idea is as follows. Initially each node has no parents. It then adds incrementally that parent whose addition results in the largest increase in the score of the resulting structure. When the addition of no single parent can increase the score, it stops adding parents to the node. The pseudo code is shown in Algorithm 9. The random variable of the sensor is fixed as the last entry by assuming it is not the parent node of any other variables, while the order of parents is random. Parameter learning is relatively simple: when the structure is learned, a maximum likelihood estimation, i.e., counting the probability of each node from the data, is used to obtain the CPD tables. The evaluation of testing using BN is just to check the probability in the CPD table. An alarm is raised if the corresponding entry in the table is equal to zero, i.e., such a concurrent event does not exist in the training data.

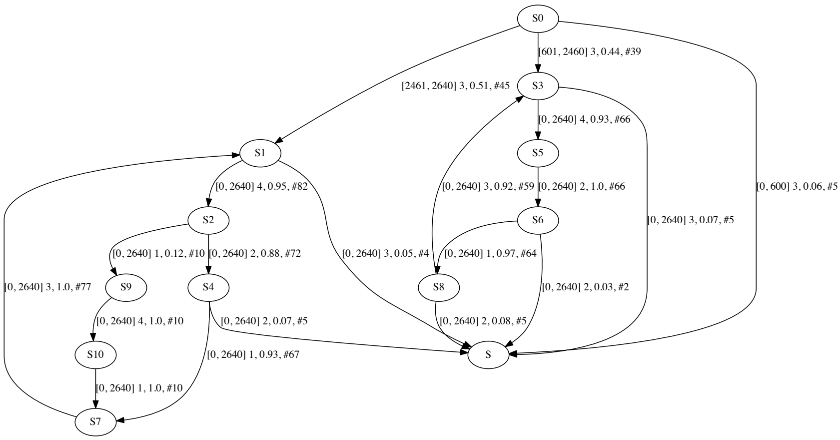


Figure 6.10: Timed automaton learned from LIT101. 1, 2, 3, 4 are symbols for QD, SC, SU, QU. S1 is the sink state, which is introduced due to the fact that some sequences in the training data have low frequencies of occurrence. The sink states are left without split or merge with other states due to lack of evidence for statistical testing.

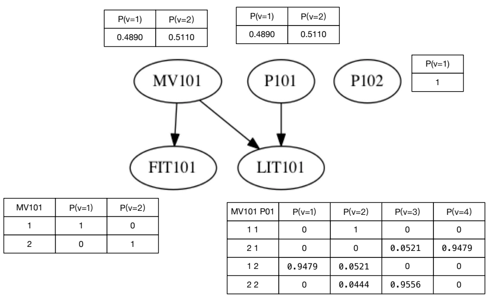


Figure 6.11: Bayesian network learned from P1. The unit of the LIT101 node in the table from the first column and the first row indicates that both MV101 and P101 are closed, so the probability that water level quickly decreases (QD) is 0. Note that the actuators' status, open and closed, are denoted as 2 and 1, respectively.

Algorithm 9 K2

Require: A set of n nodes, and ordering on the nodes, an upper bound u on the number of parents a node may have, and a dataset D containing m cases.
Ensure: Parent of each node

```
for all i := 1 to n do, do
    π_i := ∅
    P_old := f(i, π_i);
    OKToProceed := true;
    while OKToProceed and |π_i| < u do
        let z be the node in Pred(x_i) − π_i that maximized f(i, π_i ∪ {z})
        P_new := f(i, π_i ∪ {z})
        if P_new > P_old then
            P_old := P_new
            π_i := π_i ∪ {z}
        else
            OKToProceed := false
        end if
    end while
    write parents node in each node
end for
```

6.6. EXPERIMENTS

This section presents the evaluation of TABOR and discussion based on the learning experience.

6.6.1. EVALUATION

The traditional way of evaluating anomaly detection is essentially a *point-based* approach. It considers multivariate time series data at each time point as an isolated instance. However, most practical attacks in real life happen in a continuous period of time, such as the attack scenarios in the SWaT dataset that last continuously from minutes to hours. For the method we need to determine how many attack scenarios can be detected and the coverage in each detected scenario. The traditional scoring methods, such as precision and recall, do not suffice because they only look at data points instead of windows (Lavin and Ahmad, 2015). In this chapter, a novel way of evaluating anomaly detection in a CPS system is proposed by borrowing the concept of *time series discord* from the data mining community. The mining task of time series discord is actually finding abnormal subsequences in time series data (Keogh et al., 2007; Sivaraks and Ratanamahatana, 2015), which is similar to the *scenario-based* or *window-based* detection goal in this chapter. As illustrated in Figure 6.12, a false positive is a detected subsequence without an overlap between any ground-truth scenario. For the case of true positive, the coverage percentage (CP, proportion of overlap length and total ground-truth scenarios length) evaluates the quality of detection coverage, while the penalty score (PS, with time as unit) evaluates the length of detection outside the overlap. A good detection result should have a high coverage percentage (close to 1) and a small number of penalty scores.

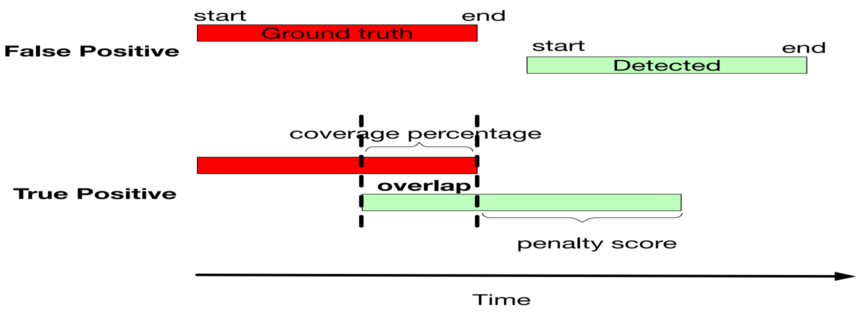


Figure 6.12: Defining true positive and false positive.

All types of alarms are listed as follows:

1. Timed automaton
   (a) Event error: an invalid event in a given state;
   (b) Timing error: an event duration outside valid timing guard;
   (c) State error: reaching a sink state, where the computation halts.
2. Bayesian network: a zero probability entry in the CPD
3. Out of alphabet (OOA): the sensor value exceeds the threshold (i.e., $min - c\sigma$ and $min - c\sigma$), and the actuator value did not occur in the training data.

In Section 6.4 it is mentioned that the sensors in SWaT are grouped based on their different properties. For the LIT and DPIT sensors, the signal shows regular patterns. The TA and the BN models are learned as shown in models 1, 5, 6, and 9 (see Table 6.1). One key question is how to fuse the results from the TA and the BN model. Considering the high cost of false positive in a large water plant, a conservative *and* strategy is used to fuse the results, i.e., only adopting alarms raised from both the TA and the BN model. However, the OOA errors are directly adopted into the final result because they tend to show obviously abnormal patterns with a high priority. Table 6.2 shows the results of using the fused results and the single result from each model. It is imaginable that using an *or* strategy will get more true positives but much more false positives. Table 6.3 presents detailed results from each model. Figure 6.13 shows an example of the result fused from different types of alarms.

For the chemical sensors AIT and pressure sensor PIT, only the deviation of the differential is checked, i.e., an OOA type. Because the resulting timed strings do not show stationary behaviors, a single symbol checking is thus deployed for anomaly detection. Examples from AIT202 and PIT501 are shown in Figure 6.14 and Figure 6.15.

To make a comprehensive comparison with existing literature, the point-based recall evaluation in each attack scenario is also conducted in this chapter, as shown in Table 6.4. Our proposed model successfully detects 24 out of 36 scenarios, while the DNN

Table 6.2: Comparison only using TA or BN

Model number	Method	PF	TP	CP (%)	PS (s)
1	TABOR	0	9	7.85	1889
1	TA	7	26	87.48	189516
1	BN	0	5	1.95	629
1	OOA	0	5	5.90	1260
5	TABOR	0	3	63.36	70
5	TA	16	13	73.95	47645
5	BN	0	4	64.03	70
5	OOA	0	0	0	0
6	TABOR	0	5	65.12	145
6	TA	3	24	81.32	188996
6	BN	0	3	63.32	33
6	OOA	0	2	1.80	112
9	TABOR	0	4	61.67	233
9	TA	0	3	60.23	856
9	BN	0	2	59.40	73
9	OOA	0	4	61.67	233

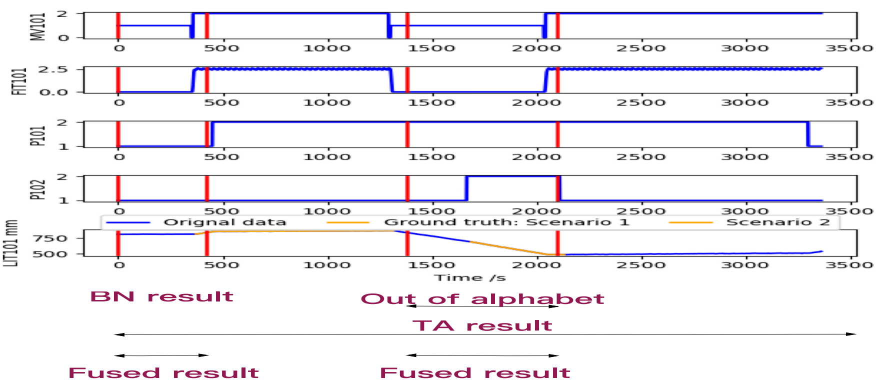


Figure 6.13: An example of fused results. The timed string of this example is (3,420)(2,960)(1,720)(3,1260), which is not fired by the TA as a whole sequence. The probability of the aligned event $P(LIT = 3)|(MV = 1, P101 = 1) = 0$. The actuator P102 is never seen to be open in the training data.

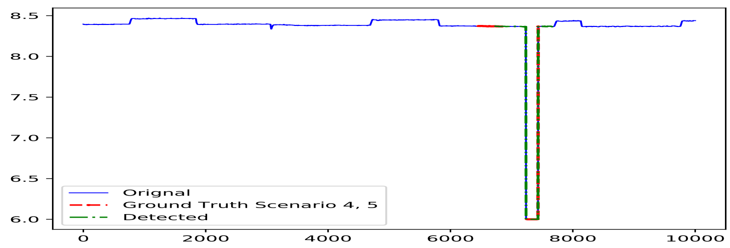


Figure 6.14: An example of the detection result from the chemical measurement sensor AIT202.

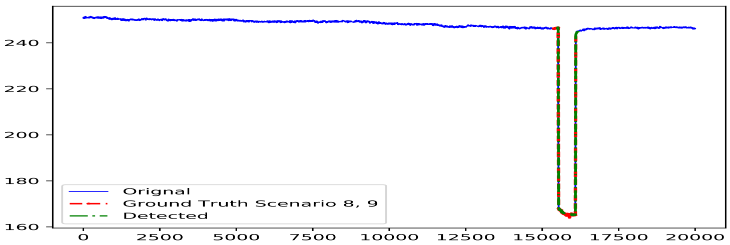


Figure 6.15: An example of detection results from the press measurement sensor PIT501.

6

Table 6.3: Results of each model

Model Number	FP	TP	Detected Scenarios	CP (%)	PS (s)
1	0	9	1, 2, 16, 21, 25, 28, 29, 30, 31	7.85	1889
2	0	0	0	0	0
3	0	2	4, 5	0.60	637
4	0	0	0	0	0
5	0	3	7, 22, 23	63.50	117
6	0	5	7, 13, 22, 23, 30	65.12	145
7	0	0	0	0	0
8	0	7	8, 9, 17, 23, 33, 34, 35	11.99	2363
9	0	4	9, 17, 23, 35	61.67	233
10	0	6	8, 9, 17, 23, 34, 35	3.22	1867
11	0	6	8, 9, 17, 23, 33, 35	5.50	1360
12	0	1	23	0.38	513
13	0	9	8, 9, 14, 15, 17, 23, 32, 34, 35	4.17	1354
14	0	6	8, 9, 17, 23, 32, 35	3.11	965
15	0	7	8, 9, 17, 23, 32, 34, 35	2.36	1127
16	0	7	8, 9, 17, 23, 32, 34, 35	4.50	877

and the SVM detect 13 and 20, respectively. The further overall comparison is shown in Table 6.5. The proposed model has slightly more false positives but a much better recall, thus the overall performance in F-measure is superior over the DNN and the SVM models with 2-3% relative improvement. Moreover, the runtime comparison in Table 6.6 shows that the computation is highly efficient in the proposed model. The key advantage is that the original multivariate signal is partitioned into groups and segmented to dramatically reduce the dimension and the size of training data. The learning of RTI+ and K2 are both polynomial time. The computation complexity of testing each event is just $O(1)$ in both the TA and BN model.

Table 6.4: Points evaluation in each scenario. The second column of scenario number is consistent with the original attacks description. However, some of them do not have any actual impact in SWaT. Basically only 36 scenarios are counted in this chapter and the literature.

NO.	NO. Scenario	Description of attack	DNN	SVM	TABOR
1	1	Open MV-101	0	0	0.049
2	2	Turn on P-102	0	0	0.930
3	3	Increase LIT-101 by 1mm every second	0	0	0
4	4	Open MV-504	0	0.035	0.328
5	6	Set value of AIT-202 as 6	0.717	0.720	0.995
6	7	Water level LIT-301 increased above HH	0	0.888	0
7	8	Set value of DPIT as >40kpa	0.927	0.919	0.612
8	10	Set value of FIT-401 as <0.7	1	0.433	0.994
9	11	Set value of FIT-401 as 0	0.978	1	0.998
10	13	Close MV-304	0	0	0
11	14	Do not let MV-303 open	0	0	0
12	16	Decrease water level LIT-301 by 1mm each second	0	0	0
13	17	Do not let MV-303 open	0	0	0.597
14	19	Set value of AIT-504 to 16 uS/cm	0.123	0.13	0.004
15	20	Set value of AIT-504 to 255 uS/cm	0.845	0.848	0.997
16	21	Keep MV-101 on continuously; Value of LIT-101 set as 700mm	0	0.0167	0.083
17	22	Stop UV-401; Value of AIT502 set as 150; Force P-501 to remain on	0.998	1	0.998
18	23	Value of DPIT-301 set to >0.4 bar; Keep MV-302 open; Keep P-602 closed	0.867	0.875	0
19	24	Turn off P-203 and P-205	0	0	0
20	25	Set value of LIT-401 as 1000; P402 is kept on	0	0.009	0
21	26	P-101 is turned on continuously; Set value of LIT-301 as 801mm	0	0	0.999
22	27	Keep P-302 on continuously; Value of LIT401 set as 600mm till 1:26:01	0	0	0.196
23	28	Close P-302	0.936	0.936	1.000
24	29	Turn on P-201; Turn on P-203; Turn on P-205	0	0	0
25	30	Turn P-101 on continuously; Turn MV-101 on continuously; Set value of LIT-101 as 700mm; P-102 started itself because LIT301 level became low	0	0.003	0.999
26	31	Set LIT-401 to less than L	0	0	0
27	32	Set LIT-301 to above HH	0	0.905	0
28	33	Set LIT-101 to above H	0	0	0.890
29	34	Turn P-101 off	0	0	0.990
30	35	Turn P-101 off; Keep P-102 off	0	0	0.258
31	36	Set LIT-101 to less than LL	0	0.119	0.889
32	37	Close P-501; Set value of FIT-502 to 1.29 at 11:18:36	1	1	0.998
33	38	Set value of AIT402 as 260; Set value of AIT502 to 260	0.923	0.927	0.996
34	39	Set value of FIT-401 as 0.5; Set value of AIT-502 as 140 mV	0.940	0	0.369
35	40	Set value of FIT-401 as 0	0.933	0.927	0.997
36	41	Decrease LIT-301 value by 0.5mm per second	0	0.357	0

6.6.2. DISCUSSION

Precise segmentation on the basis of sensor data is difficult due to the noise. Also, the classification of segments with close differential values, e.g., SU and SC in the signals from LIT101, poses a challenge to robust detection. Another more important question

Table 6.5: Points based evaluation

Method	Precision	Recall	F measure
DNN	**0.98295**	0.67847	0.80281
SVM	0.92500	0.69901	0.79628
TABOR	0.86171	**0.78803**	**0.82322**

Table 6.6: Runtime comparison

Model Number	Training	Testing
DNN	2 weeks	8 hours
SVM	30 min	10 min
TABOR	**214 s**	**33** s

is the ending time of an attack scenario. The researchers who designed the attacks in SWaT claim that the time interval between two consecutive attacks is large enough for the stabilization of the SWaT system. However, just for the LIT sensors, one cycle of water fluctuation takes more than one hour (Figure 6.6), while the shortest time difference of two consecutive attacks in the SWaT system is less than 10 minutes. An example false positive result in Model 6 detected by TA is shown in Figure 6.16. An obvious abnormal pattern is seen in the sensor signal. However, no attack actually took place. The irregular signal is caused by the stabilization following the attack scenarios 8 and 9. A better, and more fair, way of evaluation is an open question in the anomaly detection of CPS, e.g., the ending time of an attack should be on the basis of no more impact on the system rather than the end point of an attack behavior. The main reason for a false negative is the conservative result fusion strategy from the TA and the BN model. How to combine the results of different models and make a tradeoff between sensitivity and robustness are challenging problems. Dealing with concurrent attacks on a same node of TABOR is also a challenging problem. Our system is only able to separate them if different types of alarms are raised.

6

6.7. CONCLUSION AND FUTURE WORK

In this chapter, a novel graphical model-based approach is proposed to learn the local behavior of a complex water treatment plant. The model profiles the normal behavior of the SWaT system, which is further used for anomaly detection. This technique can be considered as a combination of machine learning and specification-based detection. On one hand, it provides an inexpensive and automated learning approach for specification mining from an industrial control system without the need of expert knowledge. On the other hand, the resulting specification-like model is highly interpretable and useful for the validation and the localization of abnormal sensors or actuators in the system.

We have already started working on a state-based version of TABOR, i.e., modeling

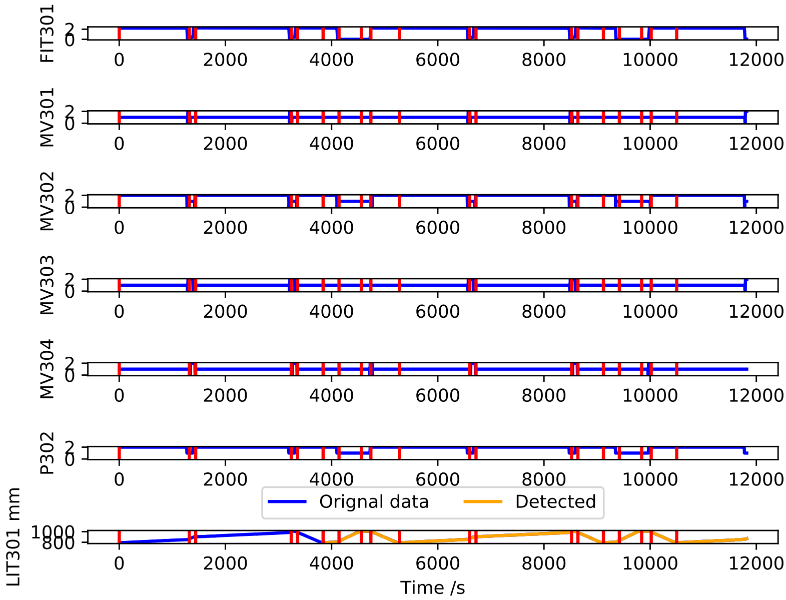


Figure 6.16: An example of detection results from PIT501. The false alarm is caused by the stabilization procedure after an attack.

6

sequential control behavior of actuators in SWaT instead of relying on the segmentation and alignment of sensor signal. This extension also aims at discovering complex concurrent events of CPS in state space without independent event assumption in the Bayesian network. In the near future, a construction-and-correct idea can be implemented in the learning procedure, e.g., any false positive and false negative examples are considered as counter examples to improve the learning result. The behavior modeling of network traffic and network attack detection are also important in the future work, because currently in the SWaT dataset, the attacker is assumed to be sitting inside the network. Last but not least, the learned model can actually be used as a simulation controller, which can be used for attack-response platforms in further research.

7

VERIFICATION OF LEARNING-BASED HYBRID CONTROL SYSTEM

The last chapter discusses the safety protection of cyber-physical systems via intrusion detection techniques. This chapter aims to formally verify the safety of intelligent control systems.

We use a hybrid model checker to explore the reachable states of a human-like cruise controller based on the MOHA model. We conclude that the pure data-driven model learned from human driving data is not guaranteed to be safe. We formally prove that the collision-avoidance can be guaranteed by adding a headway-conditioned auto-brake state.

The material in this chapter is from a draft under submission: Qin Lin and Sicco Verwer. Learning a Provably Safe Adaptive Cruise Controller from Human Driving Data (submitted)

7.1. Introduction

Adaptive cruise control (ACC) systems assist drivers to maintain safety spacing from leading vehicles and ease the workload of frequent acceleration and deceleration operations. A key drawback of existing ACCs is the inconsistency between systems and human driving habits, since the control algorithm of an ACC is based on mathematical optimization of safety and comfort rather than mimicking actual driving behaviors (Hiraoka et al., 2005).

An alternative approach is imitation learning, which mimics human control strategies in order to obtain behavior that is similar to the driving trajectories of human drivers. As a representative work, convolutional neural networks (CNNs) have been successfully applied to map raw pixels from a single front-facing camera directly to steering commands (Bojarski et al., 2016).

For such a safety-critical system, however, it is important to know whether an imitation learning cruise controller is safe to use, i.e., whether it can cause collisions or not. In (Tian et al., 2018) such a study is performed. They use simulations to test the safety properties of controllers based on deep neural networks. We argue, however, that since unexpected situations will at some point occur in practice, testing these properties in simulations is insufficient.

In Chapter 3, an imitation-learning-based model named **m**ulti-m**o**de **h**ybrid **a**utomaton (MOHA) has been proposed to mimic car-following behaviors of human drivers (Lin et al., 2018b). This model includes both *discrete observations* and *continuous output actions*. The observations are obtained by discretizing signal values such as speed and distance to the leading vehicle. The output controls the acceleration pedal of the following vehicle.

7

In this chapter, we demonstrate that the logical nature of the MOHA controller allows it to be formally verified using the SpaceEx hybrid system model checker (Frehse et al., 2011). This was recently achieved for a simplified traditional (not learned) ACC in (Mishra and Roy, 2016). To the best of our knowledge, we provide the first demonstration of formal verification for automatically learned ACCs.

The main idea of our work is to use SpaceEx to verify whether collisions are avoided by MOHA when given a non-deterministic leading vehicle. The leading vehicle is only constrained by vehicle dynamics, e.g., it can produce any trajectory falling within physically possible speed and acceleration ranges. To achieve this, we develop a transformation *MO2SX* from the discrete observations that trigger state transitions in the MOHA model to a set of linear inequalities that can be used by SpaceEx. In addition, we enhance the MOHA model to include actions for any possible future action, including those that never occurred in the training data but might be tested by the model checker.

We perform experiments in a variety of traffic for both highway and urban driving scenarios. The experiments demonstrate that purely learning a MOHA controller from data is unsafe, e.g., that it can collide in extreme cases. We then add a *safety state* to the MOHA model (a common addition to ACC systems). Essentially, the controller is forced to push the brake if the time needed to reach the current position of the leading vehicle drops below, for instance, the two seconds suggested in the highway driving scenarios. We show that:

- The MOHA controller with safety state is guaranteed to be collision-free.

- The MOHA is more safe, more accurate, and more human-like than existing controllers and neural networks.

These results demonstrate clear advantages of using explainable models based on logic (such as the MOHA) over black-box models (such as neural nets) for imitation learning. Most importantly, to the best of our knowledge, we provide the first formally verified ACC controller learned from data. Instead of trusting an AI-based system based on simulations, our work demonstrates the possibility of verifying with certainty whether an AI-based system is safe. We believe this constitutes an important step in the direction of trustworthy AI.

7.2. RELATED WORK

Verifying the safety of hybrid models is known to be undecidable except for severely restricted models such as timed automata and initialized rectangular automata (Alur et al., 1995). There exist three categories of techniques/tools that address relaxed versions of this problem.

The first category is *deductive verification*, which combines user interaction with an automated theorem prover in a proof search utilizing differential logics (Loos et al., 2013). KeYmaera is the dominating tool in this category, which has been used for safety verification of vehicle-to-vehicle (V2V) communication in ACCs (Loos et al., 2013).

The second category is *symbolic reachability analysis*, which includes tools such as HyTech (Henzinger et al., 1997a) for linear hybrid automata, d/dt (Asarin et al., 2002), PHAVER (Frehse, 2005), SpaceEx (Frehse et al., 2011) for piecewise linear affine dynamics, and Flow* (Chen et al., 2013) for non-linear dynamics. In these techniques, symbolic reachability algorithms iteratively explore reachable states starting from the initial states. There is no termination guarantee because the algorithm may reach more and more states without being able to conclude that the system is safe. In practice, setting a maximum number of states, a fix-point reaching criterion, or a maximum running time are used to force termination. In related work, a highly simplified ACC with constant acceleration and deceleration in an open-loop control system is verified using symbolic reachability analysis in SpaceEx (Mishra and Roy, 2016).

The third category is called *abstraction*. The main idea is obtaining an abstraction of coarse dynamics over the original model. Proving the safety of the abstract model then is a sufficient condition for proving the corresponding properties in the original model (Henzinger et al., 1998). The drawback is that it can be difficult to avoid an oversimplification.

In this work, we use symbolic reachability analysis using SpaceEx, similar to the work of (Mishra and Roy, 2016) but using a complex model that has been learned from data.

Related is also recent works on generation of test cases for neural networks. Deep neural networks are a popular method for learning dynamics such as those in ACCs. DeepXplore (Pei et al., 2017) and DeepTest (Tian et al., 2018) propose white-box and gray-box methods for automated generation of test cases and discovering the corner cases from a deep neural network (DNN). However, they focus more on software logic testing using a coverage criterion. This type of testing is incomplete and does not perform a full reachability analysis.

7.3. MOHA: AN HYBRID AUTOMATON MODEL

Definition 7.1. Hybrid automaton: A hybrid automaton H is a tuple $<\mathbf{Loc}, \mathbf{Edge}, \Sigma, \mathbf{X}, \mathbf{Init}, \mathbf{Inv}, \mathbf{Flow}, \mathbf{Jump}>$ where:

- **Loc** is a finite set $\{l_1, l_2, \cdots, l_m\}$ of (control) locations that represent control modes of the hybrid system, which are essentially discrete states in a finite state automaton.

- Σ is a finite set of events.

- **Edge** $\subseteq \mathbf{Loc} \times \Sigma \times \mathbf{Loc}$ is a finite set of labeled edges representing discrete changes of control modes in the hybrid system. Those changes are labeled by events from Σ.

- **X** is a finite set $\{x_1, x_2, \cdots, x_n\}$ of n-dimension real-valued variables. For example, in a standard ACC system, the variables at least include the position of the leading and following vehicles x_l and x_f, and their speeds v_l and v_f. $\dot{X}$ is for the first-order differential of variables $\{\dot{x}_1, \dot{x}_2, \cdots, \dot{x}_m\}$ inside a location. The primed variables $\{x'_1, x'_2, \cdots, x'_n\}$ are used to represent updates of variables from one control mode to another, called an assignment.

- **Init(l)** is a predicate for the valuation of free variables from X when the hybrid system starts from location l.

- **Inv(l)** is a predicate whose free variables are from X. It constraints the possible valuations for those variables when the control of the hybrid system is at location l. A commonly used convex predicate is a finite conjunction of linear inequalities, e.g. $x_1 \geq 3 \wedge 3x_2 \leq x_3 + 5/2$, which represents a polytope consisting of multiple half-spaces.

- **Flow(l)** is a predicate whose free variables are from $X \cup \dot{X}$. It states the continuous system evolution for when the control mode is in location l using a differential equation (usually ordinary differential equation, ODE).

- **Jump** is a function that assigns to each labeled edge a predicate whose free variables are from $X \cup \dot{X}$. Jump(e) states when the discrete change modeled by the event e is possible and what the variable updates are when the hybrid system makes this discrete change.

Chapter 3 introduces MOHA, a novel model for learning car-following behaviors using a hybrid automaton (Lin et al., 2018b). The main idea of learning MOHA for continuous time series data is illustrated in the flowchart shown in Figure 7.1.

First, continuous variables from time series are discretized into sequences of symbolic events. Each sequence is a complete car-following trajectory from a pair consisting of a leading vehicle and a following vehicle. The time gap between two consecutive events is encoded in order to represent time-varying behaviors, e.g., moderate/harsh braking. In this way, we obtain timed strings $\{(e^i_1, t^i_1), \cdots (e^i_j, t^i_j), \cdots, (e^i_n, t^i_n)\}$ from the i-th trajectory, where t^i_j is the time difference between discrete events e^i_j and e^i_{j-1}.

Second, as a model for the discrete dynamics, a timed automaton is learned using the RTI+ real-time identification algorithm (Verwer, 2010b). The original continuous values used to obtain the corresponding discretized values in the timed string are stored in every state.

Third, states are partitioned based on a state subsequence clustering, i.e., several states in a subsequence cluster are grouped into one block in the automaton. These blocks form the different control modes of the ACC system.

Last, numeric data reached in distinct modes are used to identify the parameters of differential equations in these modes using differential evolution algorithms (DEA).

The environmental input in the MOHA is 3-dimensional, i.e., the relative speed, the relative distance, and the following vehicle's speed. Changes to these variables may trigger discrete state and control mode transitions. After entering a new mode, the controller uses the corresponding differential equation to generate continuous acceleration/deceleration output.

These equations are linear Helly models (Helly, 1959a). The acceleration in Helly's model is a linear function combining the relative speed ($\Delta v = v_l - v_f$) and the relative distance between the headway ($\Delta x = x_l - x_f$) and the desired headway, which is defined by :

$$\dot{v_f}(t) = C_1 \cdot \Delta v(t) + C_2 \cdot (\Delta x(t) - D(t)) \tag{7.1}$$

and

$$D(t) = \alpha + \beta \cdot v_f(t) \tag{7.2}$$

where C_1, C_2, α, β, are constant parameters that need to be calibrated. The desired headway is a function of the speed of the following vehicle and a safety distance, where α, β are the corresponding weightings for those variables. Note that, compared with the original Helly model, we neglect time delays because the SpaceEx model checker does not support tracking long historical variables i.e., all computations are on-the-fly.

7.4. HYBRID MODEL CHECKER

Hybrid model verification based on reachability computation is similar in spirit to *numerical simulation*, which produces all possible trajectories one by one to check whether the system behaves properly. The obvious drawback is the fact that all possible trajectories are non-enumerable, though it has been a popular "verification" approach in several ACC design works (Eyisi et al., 2013). The reachability algorithm explores the state space in a breadth-first manner, that is, each time step all the states reachable by all possible one-step inputs from states reachable in the previous step are found. Though the computation is costly, it provides more confidence in the correctness of the system than a small number of individual simulated trajectories. In the hybrid verification problem, an over-approximation is used for the set of reachable states, and a conventional symbolic state reachability algorithm is used. By checking whether forbidden states such as collisions are reachable, the model can be guaranteed to be safe.

7.4.1. SPACEEX

SpaceEx is a powerful and popular tool for safety verification of hybrid systems. It supports hybrid systems with linear piecewise affine and non-deterministic dynamics, i.e., $\dot{\mathbf{X}} = \mathbf{AX} + \mathbf{b}$, where $\mathbf{b}$ is non-deterministic turbulence. SpaceEx consists of three main

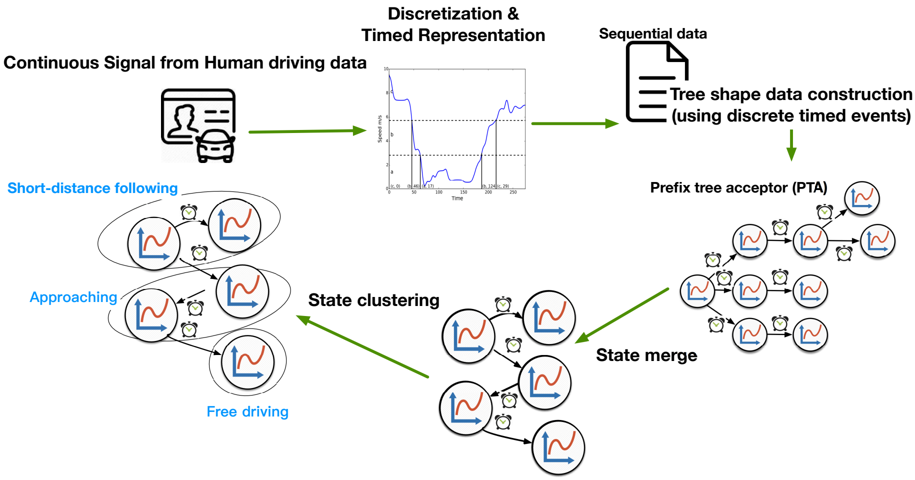


Figure 7.1: Flowchart illustrating MOHA learning. The discretization on the one-dimension signal is just for a simple demonstration. The original signal is multidimensional. Also, MOHA shows more than 3 modes in car-following behaviors (Lin et al., 2018b).

components: *Model editor* is a graphical editor for creating models of complex hybrid systems. *Analysis core* is a command line program that takes a model file in .xml format, and a configuration file .cfg that specifies the initial states. *Web interface* is a graphical user interface with which one can specify initial states and other analysis parameters, run the analysis core, and visualize the output.

7.4.2. TRANSLATOR

Though SpaceEx is becoming a user-friendly tool, the modeling is still manual. If the model under verification is complex, an automated modelling tool is needed to bypass the tedious modeling process. In our case, we intend to verify a MOHA model, consisting of a timed automaton model, parameters of continuous models in modes, and a discretization of continuous signals into discrete symbols. *MO2SX*, the translator developed in this chapter, fills the gap between MOHA and SpaceEX. Users only need to work on learning and tuning parameters of MOHA, and the output model is automatically translated to SpaceEX for safety verification. The input and output files of *MO2SX* are illustrated in Figure 7.2. MO2SX automatically obtains a SpaceEx model file with 1500 lines of code. which is burdensome for a manual modeling.

Guard linearization and model completing are two critical problems we need to address in the translating procedure, which are elaborated as follows.

GUARD LINEARIZATION

In the MOHA model, the numeric environmental input is discretized into discrete event symbols according the closest centroids in the 2-norm, i.e., $S_i = \{x_p : ||x_p - m_i||^2 \leq ||x_p - m_j||^2, \forall j, 1 \leq j \leq k\}$, where S_i is the assigned index of the centroid (symbol), x_p the numeric data, m_i, m_j centroids, and k the number of centroids. The centroids are

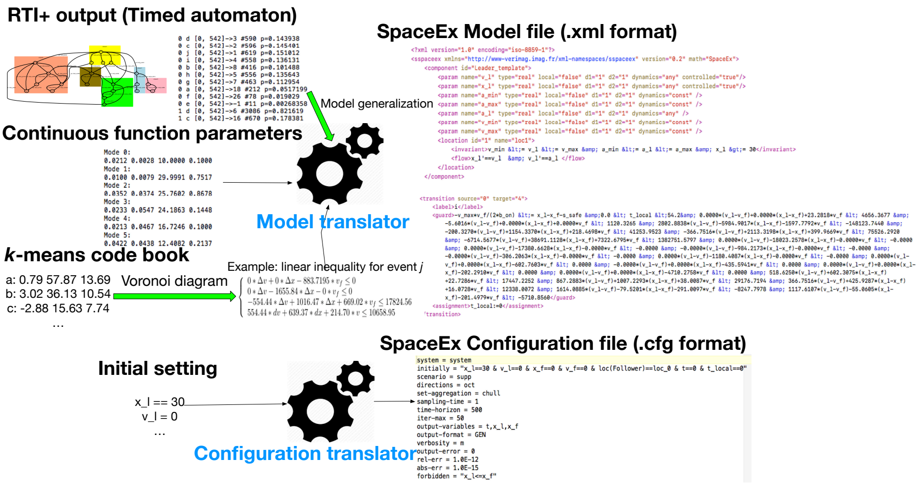


Figure 7.2: Translator MO2SX. The files on the left side are from MOHA and the initial setting. The files on the right side are supported for model checking in SpaceEx.

learned using the k-means clustering algorithm and used to trigger state transitions. This representation is non-linear and not supported by existing hybrid model checkers. To circumvent this issue, we translate the clusters into a bounded three-dimensional Voronoi Diagram (Aurenhammer, 1991). The main idea is to partition a bounded 3-d space into regions (polyhedra, the number of which is equal to the number of centroids), that are represented by linear inequalities. In each solid polyhedron, all points are closest to its own centroid.

Each polyhedron consists of several hyperplanes, i.e., a conjunction of linear inequalities, as illustrated in Figure 7.3. Note that the MOHA model shown in (Lin et al., 2018b) has 10 discrete events from “a" to “j", which are essentially symbolic representations from k-means clustering on continuous data. Therefore, 10 polyhedra are obtained by the Voronoi diagram.

MODEL GENERALIZATION

Due to the limited traffic scenarios in the training data, the learned automaton model is incomplete and does not contain a transition for every possible situation. We complete the model by adding transitions for unseen events and directing them to the initial state.

Taking $S1$ for an example as shown in Figure 7.4, the added symbols are the neighboring polyhedra of existing events “d" and “c". We obtain these by searching for adjacent polyhedra, as illustrated in Figure 7.3. We only require neighboring polyhedra because we assume that trajectories cannot jump between nonadjacent polyhedra (essentially skipping an event). We redirect new transitions to the initial state because this implements a type of recovering behavior: when the controller has no idea about what to do next (something unexpected occurs), it makes no assumptions about the past (by returning to the initial state), and assumes any future is possible.

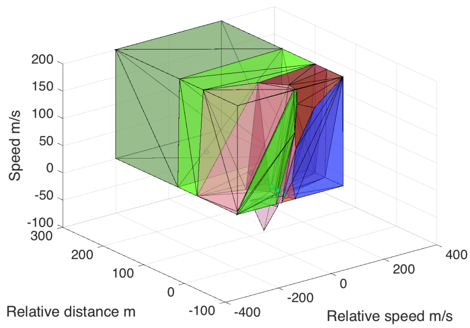


Figure 7.3: Polyhedra obtained by Voronoi diagram linearization. Discrete events are illustrated by different colors.

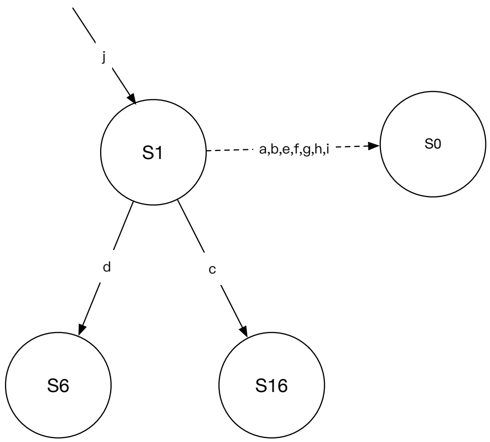


Figure 7.4: An illustrative example of completing outgoing transitions in S1 of the MOHA.

7.5. MODELING AND EXPERIMENTS

Our experimental framework (shown in Figure 7.5) consists of two components running in parallel: a nondeterministic *leading vehicle* with constraints about speed and acceleration and a *following vehicle* equipped with a cruise controller. The *autobrake* state is used for handling automatic brake scenarios when the relative distance is small. We will compare the safety performances with and without this state. In this chapter, the leading vehicles running in highway and urban traffic are studied:

- Highway: We adopt the general legitimate range on the highway: 80-120 km/h (see all settings shown in Table 7.1). The leading vehicle operates nondeterministically. Such a speed range is the working condition of a standard ACC system (NISSAN).
- Urban: We adopt the general legitimate range in the urban environment: 10-80 km/h (see all settings shown in Table 7.1). The leading vehicle conducts a nonde-

Table 7.1: Parameter settings in highway scenarios (top) and urban scenarios (bottom)

Parameters	values	Parameters	values
v_{l_min} (m/s)	22	v_{l_max} (m/s)	33
v_{f_min} (m/s)	0	v_{l_max} (m/s)	33
x_{l0} (m)	150	v_{l0} (m/s)	[22,33]
$a_{f\_max}(m/s^2)$	6	a_{f_min}, a_{l_min} (m/s^2)	-4
$a_{l\_max}(m/s^2)$	0	v_{f0} (m/s)	[22,33]
v_{l_min} (m/s)	3	v_{l_max} (m/s)	22
v_{f_min} (m/s)	0	v_{f_max} (m/s)	22
x_{l0} (m)	150	v_{l0} (m/s)	[3,22]
$a_{f\_max}(m/s^2)$	6	$a_{f\_min}(m/s^2)$	-4
$a_{l\_max}, a_{l_min}(m/s^2)$	0	v_{f0} (m/s)	[3,22]

terministic running. Such a new scenarios is for testing the generalization of the model, because the training data of the MOHA are from highway traffic.

We evaluate three different control strategies:

- Pure MOHA **(P-MOHA)**: A MOHA purely controls the following vehicle without an additional emergency brake state. We will investigate if the Pure-MOHA learned from human car-following behaviors is already safe for cruise control. The MOHA models with single mode and multiple modes are called **S-MOHA** and **M-MOHA** for short, respectively.

- Autobrake state on basis of braking distance+MOHA **(BD-MOHA)**: In existing ACCs, a warning notifies the driver to take over or (semi-)automatically switches to a braking state when the relative distance is too short. In this work, a safety state is added to the data-driven P-MOHA to deal with emergency and automatic braking scenarios. The trigger condition of the braking state is that the relative distance Δx is smaller than the braking distance $\frac{v \cdot v_{max}}{2*a_{min}}$. Note that theoretically the braking distance is $\frac{v_f^2}{2*a_{min}}$. Due to the limited support functionality of linear equations of SpaceEx, the simplified condition is used alternatively.

- Autobrake state on basis of headway-in-time+MOHA **(HIT-MOHA)**: The headway-in-time (HIT) is usually suggested in daily highway driving scenarios. The follower's desired distance is set to $v_f \times t_{headway}$ for a given $t_{headway}$., i.e., the relative distance should be greater than the distance the follower would travel in $t_{headway}$ without reducing speed.

Another motivation for setting an autobrake state is from the theoretical analysis of the minimum deceleration in the Helly model. Taking the single mode identified from the natural data with $C1 = 0.0425$, $C2 = 0.0051$, $\alpha == 22.37$, and $\beta = 0.1$ for example, in the worse case, we get $\Delta v = -33$, $v_f = 33$. The full deceleration derived from Equation 7.1 and Equation 7.2 is -1.68, which is significantly less powerful than the full deceleration -4 used in this chapter.

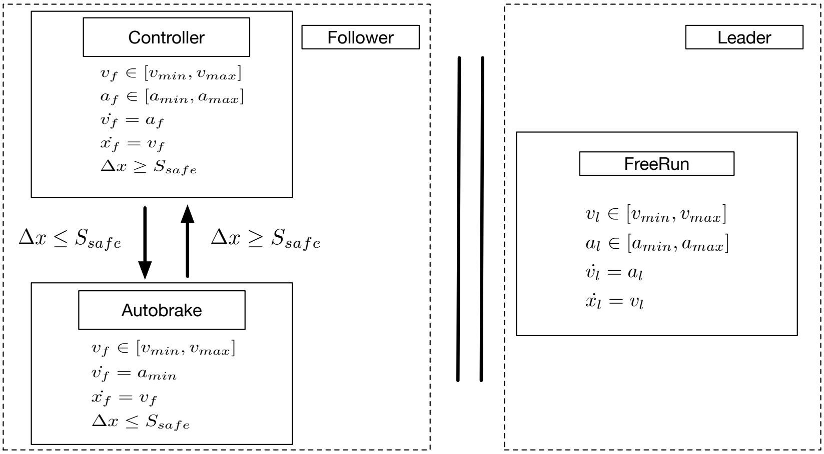


Figure 7.5: Modelling overview of the experiments.

MOHA is compared with two baseline models in this chapter. The first one is a *random follower*. A random follower with nondeterministic dynamics is an over-approximation over any controller. The proportional–integral–derivative (PID) controller is commonly used in existing ACC systems (Magdici and Althoff, 2017). Due to the limited functionalities of SpaceEx, the model checker does not allow access to long-term historical variables which are needed for the derivative part of PID. Instead, we use an auxiliary automaton as a one-step-past memory storage, the *PD controller* is implemented and serves as the second baseline with the form:

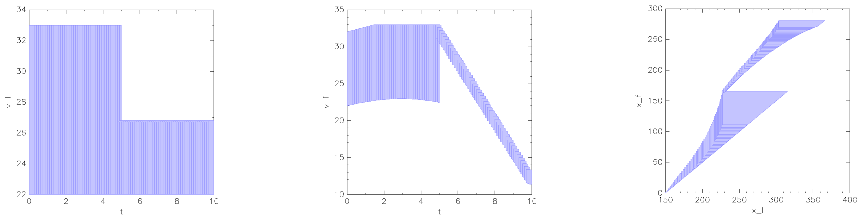
$$\begin{cases} d_{des}(i) = d_{safe} + v_f(i) \\ err = dx(i) - d_{des}(i) \\ a_{pid}(i) = k_p * err(i) + k_d * (err(i) - err(i-1)) \end{cases} \tag{7.3}$$

The parameters are well-tuned on the NGSIM dataset as $K_p = 0.8$, $K_d = 0.03$, $d_{safe} = 20m$ (Zhang et al., 2018).

The parameters of vehicle dynamics are also presented in Table 7.1. These settings are used in the literature (Zhang et al., 2018). In both cases, the following vehicle starts tracking at the maximum relative distance detectable by the ACC radar system, i.e., 150 m. The following vehicle is allowed for a standstill for testing the braking functionalities. The initial states in both cases are uncertain bounded by reasonable intervals.

An example of the reachability results in the highway scenarios of the single mode HIT-MOHA is shown in Figure 7.6. Table 7.2 summarizes the safety for all models and control strategies. It can be observed that the safety state boosts the safety of the controllers. The pure MOHA model is not guaranteed to be safe, unfortunately.

However, introducing the extra safety state potentially sacrifices the similarity to human car-following behavior. The imitation accuracy, or less formally *human likeness*, is evaluated using a test set from the NGSIM dataset. The main idea is that for each car-following episode, the trajectory of the leading vehicle and the initial status of the following vehicle are provided. The complete trajectory of the following vehicle is gener-



(a) Reachable states of x_l (m) v.s. t (s)

(b) Reachable states of x_f (m) v.s. t (s)

(c) Reachable states x_l (m) vs. x_f (m)

Figure 7.6: Reachable states of single mode HIT-MOHA in the highway scenario. x_l and x_f are position variables for the leading vehicle and the following vehicle. It can be observed that at around 5 seconds, the auto-brake state is triggered (see the linear deceleration in subfig (b). After 7 seconds, the relative speed $v_l - v_f > 0$, collision is not possible. The model checker verifies that at any state $x_l > x_f$ (cf. subfig (c)).

Table 7.2: Safety summary of all models.

Scenarios	Model	Condition	Safe?
Highway	P-MOHA	-	×
	S-MOHA	HIT	√
	M-MOHA	HIT	√
	Random	HIT	√
	PD	HIT	√
	All above	BD	√
Urban	P-MOHA	-	×
	S-MOHA	HIT	√
	M-MOHA	HIT	√
	Random	HIT	√
	PD	HIT	√
	All above	BD	√

ated using controllers and compared with the human drivers' trajectories present in the testing data. A small trajectory difference indicates a better human-likeness score. The results are presented in Table 7.3. The score is the mean square error between simulated trajectories and those of human drivers. A feed-forward neural network (FNN) is additionally compared as a baseline of imitation learning with default settings (Simonelli et al., 2009; Wang et al., 2018). Note that generating whole trajectories is essentially an iterative procedure, i.e., the trajectory at $t+1$ relies on the result at t. An additional one-step prediction is shown in Table 7.4 to demonstrate the actual predictive performance of the learned models. The difference between the results in Table 7.3 and Table 7.4 can be seen as the difference between multi-step prediction and one-step prediction.

From the results, we make the following observations:

1. Safety is not guaranteed when learning a Pure-MOHA controller. This makes sense

Table 7.3: Human likeness score comparison-multi steps

	Model	Error (m/s)	Jerk (m/s^3)
Without safety state	M-MOHA	**0.1083**	0.0037
	S-MOHA	0.1124	**0.0029**
	PD	0.1387	0.0438
	FNN	0.3451	0.0047
	Human	-	0.0574
With safety state	M-MOHA	**0.1037**	0.0373
	S-MOHA	0.1089	**0.0323**
	PD	0.1391	0.0380
	FNN	0.2411	0.0359
	Human	-	0.0574

Table 7.4: Human likeness score comparison-one step

	Model	Error (m/s)	jerk (m/s^3)
Without safety state	M-MOHA	**0.0316**	0.0033
	S-MOHA	0.0317	**0.0025**
	PD	0.0543	0.0336
	FNN	0.0408	0.0048
With safety state	M-MOHA	**0.0329**	0.0199
	S-MOHA	**0.0329**	**0.0195**
	PD	0.0488	0.0395
	FNN	0.0423	0.0469

because the training data do not contain (near) collisions. There is no way of learning this type of behavior from the available data.

2. Switching to an autobrake state boosts the safety of ACC systems such as the MOHA. Among all control strategies, the headway control (HIT) is sufficient and is suggested by us for normal driving scenarios owing to its superior balance between safety and human likeness.

3. The BD is the most conservative control strategy. Even though it guarantees a full-speed-range scenario. It is not recommended because the significant large desired relative distance leads to poor car-following performance and traffic jams.

4. Though introducing the safety state slightly deteriorates the car-following performance in one-step prediction, the general performance in whole trajectory control is not jeopardized.

5. MOHA outperforms both the PD and the FNN baselines on human likeness, also when it includes a safety state. There is a significant jump in terms of jerk (sudden braking) when the safety state is triggered.

7.6. CONCLUSION

In this chapter, a framework to automatically learn and verify a hybrid automaton-based adaptive cruise controller is proposed. The framework consists of a learning-component MOHA and a translator *MO2SX*. The MOHA shows a superior performance to human-like car-following, while MO2SX automatically translates a MOHA model for verification by the SpaceEx hybrid model checker. We demonstrate that a MOHA model learned purely from human driving data is not guaranteed to be safe (collision-free) due to the lack of emergency brake scenarios in training data. Introducing an additional safety state guarantees this safety while maintaining good human likeness scores. To the best of our knowledge, we present the first formally verified cruise control system that is learned form data.

In the near future, we will investigate more driving behaviors learning and verification, e.g., steering control. Another interesting research line is using the model checker as an oracle providing unsafe counterexamples to improve the model learning part.

8

Conclusion, reflection, and future work

In this chapter, the main contributions of this thesis and lessons learned are first wrapped up. Then several reflections about the social impact of this thesis are made. Last, several promising directions worth researching are pointed out as future work.

8.1. Conclusion

This thesis addresses some problems oriented from the motivation of solving control problems in real life. The marriage of automatic control and automata learning has a two-fold meaning: first, the rejuvenation of automata learning benefits from impactful control applications in autonomous driving and security of cyber-physical systems. Second, the manual design and modeling of control systems is becoming more and more impractical.

The identification and verification of hybrid systems are still not fully reclaimed land and are attracting researchers from computer science and automatic control. As a powerful technique to uncover the underlying logical behaviors, automata learning is indeed welcome to help us obtain insightful sequential models. The limitation of logical model is that it focuses more on low-dimensional dynamics. The identification of high-dimensional dynamics can borrow ideas of continuous dynamics' identification from the control domain. This intersection is also happening in verification: a computer-aided technique such as computational geometry is leveraged as a powerful automated tool for analyzing the complex dynamics of hybrid systems, where theoretically proving properties such as safety from control theory is difficult.

The work in this dissertation is "retro and innovative". The traditional framework used by control scientists and engineers is "first-principle design & verification", which requires lots of effort in understanding the physical properties of a system. This framework can be innovatively replaced by "automated learning & automated verification". We claim the advantage of using explainable and verifiable models such as hybrid automata without dropping the retro on understanding the system.

Hybrid systems learning

Chapters 3 & 4 advocate the metrics of learning hybrid automata. Two approaches named *composed learning* and *incline learning* are proposed. The composed learning is a three-step approach (abstraction-abstraction-refinement): first (abstraction), learning the discrete dynamics described by a conventional finite state machine; second (abstraction), further abstracting the model by grouping the states as multiple modes via similar state sub-sequences; third (refinement) learning the detailed continuous dynamics in each mode. Incline learning treats continuous values as well as symbolic values simultaneously in the automata learning procedure. The continuous values are transferred into first-order differences and used for evaluating the similarity between states. The difference between the two aforementioned approaches is how to use the continuous data. Incline learning is more compact in learning hybrid automata but it can deal with much less complex continuous dynamics than the composed learning.

The composed learning approach is applied in car-following behaviors learning. The simulation results demonstrate that this approach achieves higher accuracy on trajectory prediction compared with state-of-the-art approaches. This model is further used as a human-like cruise controller learned from human driving data. The incline model addresses general uni-variate time series prediction problems (not just work for power forecasting as discussed in Chapter 4). The prediction results are comparable with state-of-the-art approaches. The resulting models from both approaches are insightful to discover the underlying dynamics.

Learning and control for interaction with other agents

Chapter 5 integrates model predictive control with stochastic automata learning, which is leveraged to model stochastic dynamics of the uncontrolled environment. This research addresses an ego system (the system under control) control problem considering its interaction with other involved participants (OIPs). The behaviors of the OIPs are learned using probabilistic automata inference. Instead of predicting the complete maneuvers of OIPs, the probabilities of high-level future behavioral patterns (intentions) are estimated. These intentions are used as the uncontrolled input for the ego system's control. By doing so, the optimization of control considers the influences caused by OIPs.

This framework is applied in the car-following control for autonomous driving vehicles by considering the predicted lane change intention of other participating vehicles in the traffic environment. The autonomous vehicle is an ego system, while the surrounding vehicles are OIPs, of which the lane change intentions are predicted by stochastic automata. The input of car-following control is the averaging weights from the dynamics from the leading vehicle of the ego vehicle and the cut-in vehicles. The experimental results demonstrate that this framework can improve the safety of cruise control in autonomous driving.

Learning and Diagnosing for cyber attacks

Chapter 6 deals with model learning from a control system. The challenging problem is the lack of knowledge about the input and output variables. To solve this problem, the output is assumed to be sensors' behaviors, while the input is actuators' behaviors, of which the learning is on the basis of timed automata. The dependency of input and output is obtained by learning the causality among them, of which the learning is realized by Bayesian network inference.

The application of this work is in intrusion detection for safety-critical industrial control systems owing to the growing threats of cyber attacks. The physical cyber attacks falsify the reading of sensors or actuators and disrupt the state of the system. The framework proposed in called TABOR, which is an anomaly detector combining timed automata and Bayesian network learning. TABOR learns the legitimate behaviors from a water treatment system and detects deviations from this model caused by an intrusion. The experimental results show two significant advantages of TABOR: 1) This technique can be considered as a combination of machine learning and specification-based detection. On one hand, it provides an inexpensive and automated learning approach for specification mining from an industrial control system without the need for expert knowledge. On the other hand, the resulting specification-like model is highly interpretable due to its graphical-model property and useful for the validation and the localization of abnormal sensors or actuators in the system. 2) The model has superior performance on both precision and run-time over state-of-the-art models including support vector machine and deep neural networks.

SAFETY VERIFICATION OF HYBRID SYSTEMS

Chapter 7 answers the question: once a hybrid automaton model is learned from the demonstration of a teacher, how does one rigorously prove the safety property of the model in an uncertain environment? Reachability analysis is leveraged as a tool to verify the safety of the learning-based model. The intersection of an unsafe set and reachable states of the model is computed. The safety is rigorously guaranteed because the reachable states actually over-approximate the dynamics of the original model.

A state-of-the-art hybrid systems verification tool named SpaceEx is used for verifying the human-like cruise controller studied in Chapter 3 as a case study. The experimental results show that the original model directly learned from human drivers is not guaranteed to be collision-free. Adding an auto-braking state, of which the reachable condition depends on the headway, has enhanced the controller's safety.

8.2. REFLECTION

Nowadays, intelligent systems are liberating people from tedious and even dangerous work in various domains such as robotics, transportation, and power systems to name a few. This thesis aims at making these technologies more *intelligent* and more *safe* at least in the *autonomous driving* and *public infrastructure* domains. In the following, we will discuss the social impact we could bring from this thesis.

- Autonomous driving

  Some unicorns are announcing that they will introduce the massive production of AVs with high-level autonomy. We maintain a cautious attitude towards that. Deep driving intelligence and verifiable safety are the main concerns without evidence being properly solved already.

1. *Intelligent driving assistance system*

   Human beings are delicate driving controllers that can teach AVs how to drive. In this thesis, we showcase that the existing ACC systems are encountering some problems of mismatching the human being's driving habits. Our models and learning algorithms provide a solution towards a data-driven ACC mimicking car-following behaviors from massive driving data. A much more intelligent driving assistance system including complex behaviors like lane change, turning can be developed using our techniques.

2. *Safe autonomous vehicles in uncertain traffic*

   A pure data-driven AV controller could be problematic due to its dependence on high-quality training data and generalization of the model. In this thesis, we showcase the safety verification of a data-driven ACC. This work provides a method for about automatically obtaining a truthfully safe ACC learned from human driving data. Another work is about the interactions between autonomous cars and human-driving cars, which is a well-known safety challenge in the coming years. We develop a data-driven ACC system dealing with unexpected cut-in vehicles. This system will also be formally verified in the near future. We believe that these two works will enhance the safety of AVs and at some points build up public confidence in AVs.

- Public infrastructure

  The importance of public infrastructure like power and water systems controlled by industrial control systems (ICS) is self-evident. The presence of cyber components like SCADA makes ICS vulnerable to attacks. Several attacks targeting these critical infrastructures have already happened and been reported. Our technique provides a solution for protecting these infrastructures as an intrusion detection component.

1. *Efficient and safe intrusion detection* The problems of intrusion detection can be solved in a *design-oriented* approach by deriving rules (also called *invariants* in the literature (Umer et al., 2017)) governing the physical process. For instance, the design of water treatment is normally either in the form of piping and Instrumentation (P&ID) diagrams, or the control algorithms. However, tremendous legacy plants without available design diagrams exist and make the problem significantly challenging. This thesis showcase a framework for discovering the physical process from a water treatment plant without expert knowledge and designs. In the future, we expect to develop an intelligent device monitoring the operational conditions of ICS. The device automatically and efficiently learns the behavioral models of ICS. They are used as computation models for detecting any abnormal behaviors caused by attacks or system flaws. The device is able to raise alarms to notify operators or other intelligent components to take responding actions to avoid further damages.

8.3. FUTURE WORK

HYBRID SYSTEMS LEARNING

Both composed learning and incline learning rely on the discretization of continuous signals. Because the discretization is highly application-oriented, in practice, it is more reasonable to design this part as a plug-and-play component. However, it is still worthwhile to investigate a "tighter" way of discretizing the continuous space during the automata learning procedure instead of obtaining the symbolic representation in advance. The literature (Pellegrino et al., 2017a) has shown some preliminary results in this direction. The main idea is learning the guards for the continuous signal as a state split operation for significantly distinct future continuous signal. Another improvement lies in the fact that regression automata learning in this thesis still deals with univariate signals. A possible future work would be extending into the multivariate signal. A possible solution is replacing first-order difference with high-dimensional regression models in states. The state merge can be done by investigating the similarity of parameters in the regression models.

LEARNING AND CONTROL FOR INTERACTION WITH OTHER AGENTS

The interaction studied is unfortunately unidirectional, i.e., the impact of OIPs on the ego system. It would be better to also consider the impact in the other direction. A game-theoretical approach would be a solution to learn the interacting behaviors (Yan et al., 2018). In addition, as the lane change is modeled by stochastic input, it would be possible to conduct probabilistic model checking on the safety property of the controller.

LEARNING AND DIAGNOSING FOR CYBER ATTACKS

There are two ongoing researches as the follow-up to the TABOR work: 1) The evaluation of TABOR is conducted in an off-line way on the batch of the dataset. An online version of TABOR is under development aimed at raising alarms on stream signals. The new version will be embedded as a real-time detector into the SWaT system and will be tested on more scenarios of physical attacks. 2) A more neat and unified model learning on the basis of process mining is being developed. The new model learns the sequential orders from sensors and actuators in a unified Petri net model instead of relying on two models of a timed automaton and a Bayesian network in TABOR.

SAFETY VERIFICATION OF HYBRID SYSTEMS

A complete loop for learning a safety-reliable model relies on the guidance of a correcting model using counterexamples from the verification step. The safety-sound model in this thesis is still not found in a fully automated way. In future work, one possible improvement is developing a CEGAR-like (Counter Example-Guided Abstraction Refinement) system for hybrid automata learning.

BIBLIOGRAPHY

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

S. Adepu and A. Mathur. Generalized attacker and attack models for cyber physical systems. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, pages 283–292, 2016a.

S. Adepu, G. Mishra, and A. Mathur. Access control in water distribution networks: A case study. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 184–191, 2017.

Sridhar Adepu and Aditya Mathur. Using process invariants to detect cyber attacks on a water treatment system. In *IFIP International Information Security and Privacy Conference*, pages 91–104. Springer, 2016b.

Sridhar Adepu and Aditya Mathur. An investigation into the response of a water treatment system to cyber attacks. In *High Assurance Systems Engineering (HASE), 2016 IEEE 17th International Symposium on*, pages 141–148. IEEE Computer Society, 2016c.

Chuadhry Mujeeb Ahmed, Carlos Murguia, and Justin Ruths. Model-based attack detection scheme for smart water distribution networks. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 101–113. ACM, 2017.

John E Albus, RH Anderson, JM Brayer, R DeMori, H-YF Feng, SL Horowitz, B Moayer, T Pavlidis, WW Stallings, PH Swain, et al. *Syntactic pattern recognition, applications*, volume 14. Springer Science & Business Media, 2012.

Rajeev Alur. Formal verification of hybrid systems. In *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, pages 273–278. IEEE, 2011.

Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.

Dana Angluin. Inductive inference of formal languages from positive data. *Information and control*, 45(2):117–135, 1980.

Panos J Antsaklis. Intelligent control. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2001.

Panos J Antsaklis, MD Lemmon, and James A Stiver. Learning to be autonomous: Intelligent supervisory control. *Intelligent Control Systems: Theory and Applications*, pages 28–62, 1993.

Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 365–370. Springer, 2002.

Karl Johan Aström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

Franz Aurenhammer. Voronoi diagramsâa survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

Esmaeil Balal, Ruey Long Cheu, and Thompson Sarkodie-Gyan. A binary decision model for discretionary lane changing move based on fuzzy inference system. *Transportation Research Part C: Emerging Technologies*, 67:47–61, 2016.

Yoshua Bengio and Paolo Frasconi. An input output hmm architecture. In *Advances in neural information processing systems*, pages 427–434, 1995.

Huikun Bi, Tianlu Mao, Zhaoqi Wang, and Zhigang Deng. A data-driven model for lane-changing in traffic simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 149–158. Eurographics Association, 2016.

Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

EA Bossanyi. Short-term wind prediction using Kalman filters. *Wind Engineering*, 9(1): 1–8, 1985.

Afaf Bouhoute, Ismail Berrada, and Mohamed El Kamili. A formal driving behavior model for intelligent transportation systems. In *Networked Systems*, pages 298–312. Springer, 2014.

Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. Fault diagnosis using timed automata. In *International Conference on Foundations of Software Science and Computation Structures*, pages 219–233. Springer, 2005.

Bertil A Brandin and W Murray Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–342, 1994.

L Breiman, JH Friedman, R Olshen, and CJ Stone. Classification and regression trees. 1984.

Miguel Bugalho and Arlindo L Oliveira. Inference of regular languages using state merging algorithms with search. *Pattern Recognition*, 38(9):1457–1467, 2005.

Matthias Buntins, Jens-W Schicke, Frank Eggert, and Ursula Goltz. Hybrid automata as a modelling approach in the behavioural sciences. *Electronic Notes in Theoretical Computer Science*, 297:47–59, 2013.

Kenneth P Burnham and David R Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer Science & Business Media, 2002.

A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry. Attacks against process control systems: Risk assessment, detection, and response. In *ACM Symp. Inf. Comput. Commun. Security*, 2011.

A.A. Cardenas, S. Amin, and S. Sastry. Secure control: Towards survivable Cyber-Physical Systems. In *Distributed Computing Systems Workshops, 2008. ICDCS '08. 28th International Conference on*, pages 495 –500, june 2008.

Rafael C Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium on Grammatical Inference*, pages 139–152. Springer, 1994.

Defense Use Case. Analysis of the cyber attack on the ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)*, 2016.

Christos G Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

Jorge Castro and Ricard Gavalda. Learning probability distributions generated by finite-state machines. In *Topics in Grammatical Inference*, pages 113–142. Springer, 2016.

Chenyi Chen, Li Li, Jianming Hu, and Chenyao Geng. Calibration of mitsim and idm car-following model based on ngsim trajectory datasets. In *Vehicular Electronics and Safety (ICVES), 2010 IEEE International Conference on*, pages 48–53. IEEE, 2010.

Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

Edmund M Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *ATVA*, volume 11, pages 1–12. Springer, 2011.

Maxwell Clerk. On governors. *Proceedings of the Royal Society of London*, 16:270–283, 1867.

Pamel Cobb. German steel mill meltdown: Rising stakes in the internet of things, 2015. URL https://securityintelligence.com/german-steel-mill-meltdown-rising-stakes-in-the-internet-of-things/.

William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *KDD workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

Gregory F Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

Jonathan Cryer and 2008 Kung-sik Chan. *Time Series Analysis with Applications in R*. Springer, 2008.

Guglielmo D'Amico, Filippo Petroni, and Flavio Prattico. Wind speed and energy forecasting at different time scales: A nonparametric approach. *Physica A: Statistical Mechanics and its Applications*, 406:59–66, 2014.

Ioannis G Damousis, Minas C Alexiadis, John B Theocharis, and Petros S Dokopoulos. A fuzzy model for wind speed prediction and power generation in wind parks using spatial correlation. *Energy Conversion, IEEE Transactions on*, 19(2):352–361, 2004.

Ruina Dang, Fang Zhang, Jianqiang Wang, Shichun Yi, and Keqiang Li. Analysis of Chinese driver's lane change characteristic based on real vehicle tests in highway. In *The 16th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1917–1922, 2013. doi: 10.1109/ITSC.2013.6728509.

Jan G De Gooijer and Rob J Hyndman. 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473, 2006.

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

Colin De La Higuera and Franck Thollard. Identification in the limit with probability one of stochastic deterministic finite automata. In *International Colloquium on Grammatical Inference*, pages 141–156. Springer, 2000.

Colin De La Higuera, José Oncina, and Enrique Vidal. Identification of dfa: Data-dependent versus data-independent algorithms. In *International Colloquium on Grammatical Inference*, pages 313–325. Springer, 1996.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38, 1977.

Anup Doshi and Mohan Manubhai Trivedi. On the roles of eye gaze and head dynamics in predicting driver's intent to change lanes. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):453–462, 2009.

Yangliu Dou, Daiheng Ni, Zhao Wang, Jianqiang Wang, and Fengjun Yan. Strategic car-following gap model considering the effect of cut-ins from adjacent lanes. *IET Intelligent Transport Systems*, 10(10):658–665, 2016.

Sriharsha Etigowni, Dave Jing Tian, Grant Hernandez, Saman Zonouz, and Kevin Butler. Cpac: securing critical infrastructure with cyber-physical access control. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 139–152. ACM, 2016.

Emeka Eyisi, Zhenkai Zhang, Xenofon Koutsoukos, Joseph Porter, Gabor Karsai, and Janos Sztipanovits. Model-based control design and integration of cyberphysical systems: an adaptive cruise control case study. *Journal of Control Science and Engineering*, 2013:1, 2013.

Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

David F Findley. Counterexamples to parsimony and bic. *Annals of the Institute of Statistical Mathematics*, 43(3):505–514, 1991.

Gene F Franklin, J David Powell, Abbas Emami-Naeini, and J David Powell. *Feedback control of dynamic systems*, volume 3. Addison-Wesley Reading, MA, 1994.

Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *International workshop on hybrid systems: computation and control*, pages 258–273. Springer, 2005.

Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer, 2011.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

King-Sun Fu. Learning control systems review and outlook. *IEEE transactions on Automatic Control*, 15(2):210–221, 1970.

King Sun Fu. Introduction to syntactic pattern recognition. In *Syntactic pattern recognition, applications*, pages 1–30. Springer, 1977.

Vijay Gadepally, Ashok Krishnamurthy, and Umit Ozguner. A framework for estimating driver decisions near intersections. *IEEE Transactions on Intelligent Transportation Systems*, 15(2):637–646, 2014.

Wei Gao and Thomas H Morris. On cyber attacks and signature based intrusion detection for modbus based industrial control systems. *The Journal of Digital Forensics, Security and Law: JDFSL*, 9(1):37, 2014.

Pedro Garcia, Antonio Cano, and José Ruiz. A comparative study of two algorithms for automata identification. In *International Colloquium on Grammatical Inference*, pages 115–126. Springer, 2000.

Denos C Gazis, Robert Herman, and Richard W Rothery. Nonlinear follow-the-leader models of traffic flow. *Operations Research*, 9(4):545–567, 1961.

Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Artificial Neural NetworksâICANN 2001*, pages 669–676. Springer, 2001.

C Lee Giles, Steve Lawrence, and Ah Chung Tsoi. Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine learning*, 44(1-2):161–183, 2001.

Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.

Antoine Girard, Colas Le Guernic, and Oded Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *International Workshop on Hybrid Systems: Computation and Control*, pages 257–271. Springer, 2006.

Jonathan Goh, Sridhar Adepu, Marcus Tan, and Zi Shan Lee. Anomaly detection in cyber physical systems using recurrent neural networks. In *High Assurance Systems Engineering (HASE), 2017 IEEE 18th International Symposium on*, pages 140–145. IEEE, 2017.

E Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

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

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

Cyril Goutte, Peter Toft, Egill Rostrup, Finn Å Nielsen, and Lars Kai Hansen. On clustering fMRI time series. *NeuroImage*, 9(3):298–310, 1999.

Peter D Grünwald. *The minimum description length principle.* MIT press, 2007.

Huajie Gu, Jun Wang, Qin Lin, and Qi Gong. Automatic contour-based road network design for optimized wind farm micrositing. *IEEE Transactions on Sustainable Energy*, 6(1):281–289, 2015.

Zhenhai Guo, Weigang Zhao, Haiyan Lu, and Jianzhou Wang. Multi-step forecasting for wind speed using a modified EMD-based artificial neural network model. *Renewable Energy*, 37(1):241–249, 2012.

D. Hadžiosmanović, R. Sommer, E. Zambon, and Pieter H. Hartel. Through the eye of the PLC: Semantic security monitoring for industrial processes. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 126–135, New York, NY, USA, 2014. ACM.

Samer Hamdar, Martin Treiber, Hani Mahmassani, and Arne Kesting. Modeling driver behavior as sequential risk-taking task. *Transportation Research Record: Journal of the Transportation Research Board*, (2088):208–217, 2008.

Christian Albert Hammerschmidt, Sicco Verwer, Qin Lin, and Radu State. Interpreting finite automata for sequential data. In *Interpretable ML for Complex Systems NIPS 2016 Workshop*, 2016.

Walter Helly. Simulation of bottlenecks in single-lane traffic flow. In *Proceedings of the Symposium on Theory of Traffic Flow*, pages 207–238. New York: Elsevier, 1959a.

Walter Helly. Simulation of bottlenecks in single-lane traffic flow. In *Proceedings of the Symposium on Theory of Traffic Flow*, pages 207–238. New York: Elsevier, 1959b.

Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1 (1-2):110–122, 1997a.

Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. In *International Conference on Computer Aided Verification*, pages 460–463. Springer, 1997b.

Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE transactions on automatic control*, 43(4):540–554, 1998.

Marijn JH Heule and Sicco Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 18(4):825–856, 2013.

Bryan Higgs and Montasir Abbas. Segmentation and clustering of car-following behavior: recognition of driving patterns. *IEEE Transactions on Intelligent Transportation Systems*, 16(1):81–90, 2015.

Toshihiro Hiraoka, Taketoshi Kunimatsu, Osamu Nishihara, and Hiromitsu Kumamoto. Modeling of driver following behavior based on minimum-jerk theory. In *Proc. 12th World Congress ITS*, 2005.

Daniel S Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)*, 24(4):664–675, 1977.

Serge Hoogendoorn, Saskia Ossen, and Marco Schreuder. Empirics of multianticipative car-following behavior. *Transportation Research Record: Journal of the Transportation Research Board*, (1965):112–120, 2006.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321455363.

Bo-Jian Hou and Zhi-Hua Zhou. Learning with interpretable structure from rnn. *arXiv preprint arXiv:1810.10708*, 2018.

Haijing Hou, Lisheng Jin, Qingning Niu, Yuqin Sun, and Meng Lu. Driver intention recognition method using continuous hidden markov model. *International Journal of Computational Intelligence Systems*, 4(3):386–393, 2011.

ics-cert. `https://ics-cert.us-cert.gov/`.

icsCERTAdvisory. ICS-CERT Advisories `https://ics-cert.us-cert.gov/advisories`.

Jun Inoue, Yoriyuki Yamagata, Yuqi Chen, Christopher M Poskitt, and Jun Sun. Anomaly detection for a water treatment system using unsupervised machine learning. *arXiv preprint arXiv:1709.05342*, 2017.

Goh J., Adepu S., Junejo K. N, and Mathur A. A Dataset to Support Research in the Design of Secure Water Treatment Systems. In *The 11th International Conference on Critical Information Infrastructures Security (CRITIS)*, pages 1–13, New York, USA, October 2016. Springer.

Austin Jones, Zhaodan Kong, and Calin Belta. Anomaly detection in cyber-physical systems: A formal methods approach. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 848–853. IEEE, IEEE Computer Society, 2014.

Hugues Juillé and Jordan B Pollack. A stochastic search approach to grammar induction. In *International Colloquium on Grammatical Inference*, pages 126–137. Springer, 1998.

Khurum Nazir Junejo and Jonathan Goh. Behaviour-based attack detection and classification in cyber physical systems using machine learning. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 34–43. ACM, 2016.

Md Abdus Samad Kamal, Shun Taguchi, and Takayoshi Yoshimura. Efficient vehicle driving on multi-lane roads using model predictive control under a connected vehicle environment. In *Intelligent Vehicles Symposium (IV)*, pages 736–741. IEEE, 2015.

Eunsuk Kang, Sridhar Adepu, Daniel Jackson, and Aditya P Mathur. Model-based security analysis of a water treatment system. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 22–28. ACM, 2016.

Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 289–296. IEEE, 2001.

Eamonn Keogh, Jessica Lin, Sang-Hee Lee, and Helga Van Herle. Finding the most unusual time series subsequence: algorithms and applications. *Knowledge and Information Systems*, 11(1):1–27, 2007.

Arne Kesting and Martin Treiber. Calibrating car-following models by using trajectory data: Methodological study. *Transportation Research Record: Journal of the Transportation Research Board*, (2088):148–156, 2008.

Edouard Klein, Matthieu Geist, Bilal Piot, and Olivier Pietquin. Inverse reinforcement learning through structured classification. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1007–1015. 2012.

Yakup Koç, Martijn Warnier, Piet Van Mieghem, Robert E Kooij, and Frances MT Brazier. The impact of the topology on cascading failures in a power grid model. *Physica A: Statistical Mechanics and its Applications*, 402:169–179, 2014.

Wolfgang Kühn. Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing*, 61(1):47–67, 1998.

Puneet Kumar, Mathias Perrollaz, Stéphanie Lefevre, and Christian Laugier. Learning-based approach for online lane change intention prediction. In *Intelligent Vehicles Symposium (IV)*, pages 797–802. IEEE, 2013.

Kevin J Lang. Random dfa's can be approximately learned from sparse uniform examples. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 45–52. ACM, 1992.

Kevin J Lang. Faster algorithms for finding minimal consistent dfas. Technical report, Technical report, NEC Research Institute, 4 Independence Way Princeton, NJ 08540, 1999.

Kevin J Lang, Barak A Pearlmutter, and Rodney A Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, pages 1–12. Springer, 1998.

Alexander Lavin and Subutai Ahmad. Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark. In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*, pages 38–44. IEEE, 2015.

Edward A. Lee. Cyber physical systems: Design challenges, http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html.

Jinwoo Lee, Minju Park, and Hwasoo Yeo. A probability model for discretionary lane changes in highways. *KSCE Journal of Civil Engineering*, 20(7):2938–2946, 2016.

Guofa Li, Shengbo Eben Li, Yuan Liao, Wenjun Wang, Bo Cheng, and Fang Chen. Lane change maneuver recognition via vehicle state and driver operation signalsâresults from naturalistic driving data. In *Intelligent Vehicles Symposium (IV)*, pages 865–870. IEEE, 2015.

Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007.

Qin Lin and Jun Wang. Vertically correlated echelon model for the interpolation of missing wind speed data. *IEEE Transactions on Sustainable Energy*, 5(3):804–812, 2014.

Qin Lin, Jun Wang, and Weiting Qiao. Denoising of wind speed data by wavelet thresholding. In *Chinese Automation Congress (CAC), 2013*, pages 518–521. IEEE, 2013.

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

Qin Lin, Sridha Adepu, Sicco Verwer, and Aditya Mathur. Tabor: A graphical model-based approach for anomaly detection in industrial control systems. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 525–536. ACM, 2018a.

Qin Lin, Yihuan Zhang, Sicco Verwer, and Jun Wang. Moha: a multi-mode hybrid automaton model for learning car-following behaviors. *IEEE Transactions on Intelligent Transportation Systems*, (99):1–8, 2018b.

Robert Lipovsky. New wave of cyberattacks against Ukrainian power industry, January 2016. http://www.welivesecurity.com/2016/01/11.

Kai Liu, Jianwei Gong, Arda Kurt, Huiyan Chen, and Umit Ozguner. A model predictive-based approach for longitudinal control in autonomous driving with lateral interruptions. In *Intelligent Vehicles Symposium (IV)*, pages 359–364. IEEE, 2017a.

Xiaoran Liu, Qin Lin, Sicco Verwer, and Dmitri Jarnikov. Anomaly detection in a digital video broadcasting system using timed automata. In *Thirty-Second Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) Workshop on Learning and Automata (LearnAut)*, 2017b.

Y. Liu, P. Ning, and M. Reiter. False data injection attacks against state estimation in electric power grids. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 21–32, 2009.

Yao Liu, Peng Ning, and Michael K Reiter. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):13, 2011.

Sarah M Loos, David Witmer, Peter Steenkiste, and André Platzer. Efficiency analysis of formally verified adaptive cruise controllers. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pages 1565–1570. IEEE, 2013.

Harry Lum and Jerry A Reagan. Interactive highway safety design model: accident predictive module. *Public Roads*, 58(3), 1995.

Daniel L Ly and Hod Lipson. Learning symbolic representations of hybrid dynamical systems. *Journal of Machine Learning Research*, 13(Dec):3585–3618, 2012.

Iain L MacDonald and Walter Zucchini. *Hidden Markov and other models for discrete-valued time series*, volume 110. CRC Press, 1997.

Silvia Magdici and Matthias Althoff. Adaptive cruise control with safety guarantees for autonomous vehicles. *IFAC-PapersOnLine*, 50(1):5774–5781, 2017.

Alexander Maier, Asmir Vodencarevic, Oliver Niggemann, Roman Just, and Michael Jaeger. Anomaly detection in production plants using timed automata. In *8th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 363–369, 2011.

Oded Maler. Algorithmic verification of continuous and hybrid systems. *arXiv preprint arXiv:1403.0952*, 2014.

João Martins, Armando Pires, A Dente, and R Vilela Mendes. Formal language control of induction motor drives. In *IEEE 2002 28th Annual Conference of the Industrial Electronics Society. IECON 02*, volume 3, pages 1903–1908. IEEE, 2002.

João F Martins, JA Dente, AJ Pires, and R Vilela Mendes. Language identification of controlled systems: Modeling, control, and anomaly detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(2):234–242, 2001.

Ramy Medhat, S Ramesh, Borzoo Bonakdarpour, and Sebastian Fischmeister. A framework for mining hybrid automata from input/output traces. In *Proceedings of the 12th International Conference on Embedded Software*, pages 177–186. IEEE Press, 2015.

Daniel Meyer-Delius, Christian Plagemann, and Wolfram Burgard. Probabilistic situation recognition for vehicular traffic scenarios. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 459–464. IEEE, 2009.

Ambuj Mishra and Subir K Roy. Towards formal verification of adaptive cruise controller using spaceex. In *VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), 2016 International Conference on*, pages 1–6. IEEE, 2016.

MA Mohandes, TO Halawani, S Rehman, and Ahmed A Hussain. Support vector machines for wind speed prediction. *Renewable Energy*, 29(6):939–947, 2004.

Brendan Morris, Anup Doshi, and Mohan Trivedi. Lane change intent prediction for driver assistance: On-road design and evaluation. In *Intelligent Vehicles Symposium (IV)*, pages 895–901. IEEE, 2011.

Maximilian Mühlegg, Florian Holzapfel, and Girish Chowdhary. Trusting learning based adaptive flight control algorithms. In *2015 AAAI Fall Symposium Series*, 2015.

John Mulder, Moses Schwartz, Michael Berg, Jonathan Roger Van Houten, Jorge Mario, Michael Aaron King Urrea, Abraham Anthony Clements, and Joshua Jacob. Weaselboard: Zero-day exploit detection for Programmable Logic Controllers. Technical report, tech. report SAND2013-8274, Sandia National Laboratories, 2013.

NGSIM. U.S. Department of Transportation, NGSIM - Next generation simulation. http://www.ngsim.fhwa.dot.gov, 2007.

Karim Nice. How cruise control systems work. <https://auto.howstuffworks.com/cruise-control.htm, 2001.

Jianqiang Nie, Jian Zhang, Xia Wan, Wanting Ding, and Bin Ran. Modeling of decision-making behavior for discretionary lane-changing execution. In *The 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 707–712. IEEE, 2016.

Oliver Niggemann, Benno Stein, Asmir Vodencarevic, Alexander Maier, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. In *AAAI*, volume 2, pages 1083–1090, 2012.

Ronald Nippold and Peter Wagner. Calibration of car-following models with single-and multi-step approaches. In *Proceedings of the Winter Simulation Conference*, page 410. Winter Simulation Conference, 2012.

NISSAN. Nissan intelligent cruise control (with low-speed following capability). https://www.nissan-global.com/EN/DOCUMENT/PDF/TECHNOLOGY/TECHNICAL/intelligent_en.pdf.

Arlindo L Oliveira and Stephen Edwards. Limits of exact algorithms for inference of minimum size finite state machines. In *International Workshop on Algorithmic Learning Theory*, pages 59–66. Springer, 1996.

Arlindo L Oliveira and João PM Silva. Efficient algorithms for the inference of minimum size dfas. *Machine Learning*, 44(1-2):93–119, 2001.

Paul Oman and Matthew Phillips. Intrusion detection and event monitoring in scada networks. *Critical Infrastructure Protection*, pages 161–173, 2007.

Christian W Omlin and C Lee Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM (JACM)*, 43(6):937–972, 1996.

José Oncina and Pedro Garcia. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*, pages 49–61. World Scientific, 1992.

Simone Paoletti, Aleksandar Lj Juloski, Giancarlo Ferrari-Trecate, and René Vidal. Identification of hybrid systems a tutorial. *European journal of control*, 13(2-3):242–260, 2007.

Margie Peden, Richard Scurfield, David Sleet, Dinesh Mohan, Adnan A Hyder, Eva Jarawan, Colin D Mathers, et al. World report on road traffic injury prevention, 2004.

Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

Gaetano Pellegrino, Christian Hammerschmidt, Qin Lin, and Sicco Verwer. Learning deterministic finite automata from infinite alphabets. In *International Conference on Grammatical Inference*, pages 120–131, 2017a.

Gaetano Pellegrino, Qin Lin, Christian Hammerschmidt, and Sicco Verwer. Learning behavioral fingerprints from netflows using timed automata. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pages 308–316. IEEE, 2017b.

Alex Pentland and Andrew Liu. Modeling and prediction of human behavior. *Neural Computation*, 11(1):229–242, 1999.

Pierre Pinson et al. Wind energy: Forecasting challenges for its operational management. *Statistical Science*, 28(4):564–585, 2013.

Louis A Pipes. An operational analysis of traffic dynamics. *Journal of Applied Physics*, 24 (3):274–281, 1953.

Jan Willem Polderman and Jan C Willems. Introduction to the mathematical theory of systems and control. *New York*, 434, 1998.

Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989a.

Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989b.

Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

Rajesh Rajamani. Adaptive cruise control. *Encyclopedia of Systems and Control*, pages 13–19, 2015.

Carl Edward Rasmussen and Zoubin Ghahramani. Occam's razor. In *Advances in neural information processing systems*, pages 294–300, 2001.

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006.

Tobias Rehder, Wolfgang Muenst, Lawrence Louis, and Dieter Schramm. Learning lane change intentions through lane contentedness estimation from demonstrated driving. In *The 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 893–898. IEEE, 2016.

Ahmet D Sahin and Zekai Sen. First-order markov chain approach to wind speed modelling. *Journal of Wind Engineering and Industrial Aerodynamics*, 89(3):263–269, 2001.

Yasubumi Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185(1):15–45, 1997.

Stan Salvador and Philip Chan. Learning states and rules for detecting anomalies in time series. *Applied Intelligence*, 23(3):241–255, 2005a.

Stan Salvador and Philip Chan. Learning states and rules for detecting anomalies in time series. *Applied Intelligence*, 23(3):241–255, 2005b.

Pedro Santana, Spencer Lane, Eric Timmons, Brian Williams, and Carlos Forster. Learning hybrid models with guarded transitions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1847–1853. AAAI Press, 2015.

S Paul Sathiyan, S Suresh Kumar, and A Immanuel Selvakumar. Optimised fuzzy controller for improved comfort level during transitions in cruise and adaptive cruise control vehicles. In *The 2015 International Conference on Signal Processing And Communication Engineering Systems (SPACES)*, pages 86–91. IEEE, 2015.

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

Roman Schmied, Harald Waschl, and Luigi del Re. Extension and experimental validation of fuel efficient predictive adaptive cruise control. In *American Control Conference (ACC)*, pages 4753–4758. IEEE, 2015.

Roman Schmied, Dominik Moser, Harald Waschl, and Luigi del Re. Scenario model predictive control for robust adaptive cruise control in multi-vehicle traffic situations. In *Intelligent Vehicles Symposium (IV)*, pages 802–807. IEEE, 2016.

Matthias Schreier, Volker Willert, and Jürgen Adamy. An integrated approach to maneuver-based trajectory prediction and criticality assessment in arbitrary road environments. *IEEE Transactions on Intelligent Transportation Systems*, 17(10):2751–2766, 2016.

Anke Schwarze, Matthias Buntins, Jens Schicke-Uffmann, Ursula Goltz, and Frank Eggert. Modelling driving behaviour using hybrid automata. *IET Intelligent Transport Systems*, 7(2):251–256, 2013.

Fulvio Simonelli, Gennaro Nicola Bifulco, Valerio De Martinis, and Vincenzo Punzo. Human-like adaptive cruise control systems through a learning machine approach. In *Applications of Soft Computing*, pages 240–249. Springer, 2009.

Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

Haemwaan Sivaraks and Chotirat Ann Ratanamahatana. Robust and accurate anomaly detection in ecg artifacts using time series motif discovery. *Computational and mathematical methods in medicine*, 2015, 2015.

Jill Slay and Michael Miller. Lessons learned from the maroochy water breach. In *International Conference on Critical Infrastructure Protection*, pages 73–82. Springer, 2007.

Padhraic Smyth. Clustering sequences with hidden markov models. In *Advances in Neural Information Processing Systems*, pages 648–654. MIT Press, 1997.

John A. Stankovic. Research directions for cyber physical systems in wireless and mobile healthcare. *ACM Trans. Cyber-Phys. Syst.*, pages 1:1–1:12, November 2016.

Andrew Stevenson and James R Cordy. A survey of grammatical inference in software engineering. *Science of Computer Programming*, 96:444–459, 2014.

Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4): 341–359, 1997.

Keith Stouffer, Joe Falco, and Karen Scarfone. Guide to industrial control systems (ICS) security. Technical Report 11, NIST special publication, 2011. URL http://csrc.nist.gov/publications/nistpubs/800-82/SP800-82-final.pdf.

Thomas A. Sudkamp. *Languages and Machines: an introduction to the theory of computer science*. Addison-Wesley, third edition, 2006.

Adam Summerville, Joseph Osborn, and Michael Mateas. Charda: causal hybrid automata recovery via dynamic analysis. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 2800–2806. AAAI Press, 2017.

Keshuang Tang, Shengfa Zhu, Yanqing Xu, and Fen Wang. Modeling drivers' dynamic decision-making behavior during the phase transition period: An analytical approach based on hidden markov model theory. *IEEE Transactions on Intelligent Transportation Systems*, 17(1):206–214, 2016.

Christian Thiemann, Martin Treiber, and Arne Kesting. Estimating acceleration and lane-changing dynamics from next generation simulation trajectory data. *Transportation Research Record: Journal of the Transportation Research Board*, (2088):90–101, 2008.

Franck Thollard, Pierre Dupont, Colin de la Higuera, et al. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *ICML*, pages 975–982, 2000.

Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, pages 303–314. ACM, 2018.

Jose Luis Torres, Almudena Garcia, Marian De Blas, and Adolfo De Francisco. Forecast of hourly average wind speed with ARMA models in Navarre (Spain). *Solar Energy*, 79 (1):65–77, 2005.

B Trakhtenbrot and YM Barzdin. Finite automate: behaviour and synthesis. 1973.

Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805, 2000.

Stavros Tripakis. Fault diagnosis for timed automata. In *FTRTFT*, volume 2469, pages 205–224. Springer, 2002.

A Troncoso, S Salcedo-Sanz, C Casanova-Mateo, JC Riquelme, and L Prieto. Local models-based regression trees for very short-term wind speed prediction. *Renewable Energy*, 81:589–598, 2015.

Muhammad Azmi Umer, Aditya Mathur, Khurum Nazir Junejo, and Sridhar Adepu. Integrating design and data centric approaches to generate invariants for distributed attack detection. In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 131–136. ACM, 2017.

David I. Urbina, Jairo A. Giraldo, Alvaro A. Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. Limiting the impact of stealthy attacks on industrial control systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1092–1105, 2016. ISBN 978-1-4503-4139-4.

Akira Ushioda. Hierarchical clustering of words. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 1159–1162. Association for Computational Linguistics, 1996.

CPIJ Van Hinsbergen, WJ Schakel, VL Knoop, JWC van Lint, and SP Hoogendoorn. A general framework for calibrating and comparing car-following models. *Transportmetrica A: Transport Science*, 11(5):420–440, 2015.

Sicco Verwer and Christian A Hammerschmidt. flexfringe: a passive automaton learning package. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*, pages 638–642. IEEE, 2017.

Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In *International Colloquium on Grammatical Inference*, pages 203–216. Springer Berlin Heidelberg, 2010a.

Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In *Grammatical Inference: Theoretical Results and Applications*, pages 203–216. Springer, 2010b.

Sicco Verwer, Mathijs De Weerdt, and Cees Witteveen. Learning driving behavior by timed syntactic pattern recognition. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1529–1534. IJCAI/AAAI, 2011.

Sicco Verwer, Rémi Eyraud, and Colin De La Higuera. Pautomac: a probabilistic automata and hidden markov models learning competition. *Machine learning*, 96(1-2): 129–154, 2014.

Sicco E Verwer, Mathijs M De Weerdt, and Cees Witteveen. Identifying an automaton model for timed data. In *Benelearn 2006: Proceedings of the 15th Annual Machine*

Learning Conference of Belgium and the Netherlands, Ghent, Belgium, 11-12 May 2006, 2006.

Sicco Ewout Verwer. *Efficient identification of timed automata: theory and practice*. PhD thesis, Delft University of Technology, 2010a.

Sicco Ewout Verwer. *Efficient identification of timed automata: Theory and practice*. PhD thesis, TU Delft, Delft University of Technology, 2010b.

Asmir Vodenčarević, Hans Kleine Büning, Oliver Niggemann, and Alexander Maier. Using behavior models for anomaly detection in hybrid systems. In *Information, Communication and Automation Technologies (ICAT), 2011 XXIII International Symposium on*, pages 1–8. IEEE, 2011.

Xiao Wang, Rui Jiang, Li Li, Yilun Lin, Xinhu Zheng, and Fei-Yue Wang. Capturing car-following behaviors by deep learning. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):910–920, 2018.

Xipeng Wang, Yi Lu Murphey, and Dev S Kochhar. Mts-deepnet for lane change prediction. In *The International Joint Conference on Neural Networks (IJCNN)*, pages 4571–4578. IEEE, 2016.

Sharon Weinberger. Computer security: Is this the start of cyberwarfare? *Nature*, 174, June 2011.

K. Wilhoit and S. Hara. The real world evaluation of cyber-attacks against ICS system. In *Society of Instrument and Control Engineers of Japan (SICE), 2015 54th Annual Conference of the*, pages 977–979, July 2015.

Samuel S Wilks. The large-sample distribution of the likelihood ratio for testing composite hypotheses. *The Annals of Mathematical Statistics*, 9(1):60–62, 1938.

Hanwool Woo, Yonghoon Ji, Hitoshi Kono, Yusuke Tamura, Yasuhide Kuroda, Takashi Sugano, Yasunori Yamamoto, Atsushi Yamashita, and Hajime Asama. Dynamic potential-model-based feature for lane change prediction. In *The Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics*, 2016.

Fei Yan, Mark Eilers, Andreas Lüdtke, and Martin Baumann. Developing a model of driver's uncertainty in lane change situations for trustworthy lane change decision aid systems. In *Intelligent Vehicles Symposium (IV)*, pages 406–411. IEEE, 2016.

Zhihai Yan, Jun Wang, and Yihuan Zhang. A game-theoretical approach to driving decision making in highway scenarios. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1221–1226. IEEE, 2018.

Y. Zhang, Qin Lin, Jun Wang, Sicco Verwer, and John Dolan. Lane-change intention estimation for car-following control in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 3(3):276–286, 2018.

Yihuan Zhang, Qin Lin, Jun Wang, and Sicco Verwer. Car-following behavior model learning using timed automata. *IFAC-PapersOnLine*, 50(1):2353–2358, 2017a.

Yihuan Zhang, Jun Wang, Qin Lin, Sicco Verwer, and John M Dolan. A data-driven behavior generation algorithm in car-following scenarios. In *Dynamics of Vehicles on Roads and Tracks Vol 1: Proceedings of the 25th International Symposium on Dynamics of Vehicles on Roads and Tracks (IAVSD 2017)*, page 227. CRC Press, 2017b.

Xi Zheng and Christine Julien. Verification and validation in cyber physical systems: research challenges and a way forward. In *Proceedings of the First International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 15–18. IEEE Press, 2015.

Song-Chun Zhu, David Mumford, et al. A stochastic grammar of images. *Foundations and Trends® in Computer Graphics and Vision*, 2(4):259–362, 2007.

SUMMARY

Automatic control is a technique about designing control devices for controlling machinery processes without human intervention. However, devising controllers using conventional control theory requires first principle design on the basis of the full understanding of the environment and the plant, which is infeasible for complex control tasks such as driving in highly uncertain traffic environment. Intelligent control offers new opportunities about deriving the control policy of human beings by mimicking our control behaviors from demonstrations. In this thesis, we focus on intelligent control techniques from two aspects: (1) how to learn control policy from supervisors with the available demonstration data; (2) how to verify the controller learned from data will safely control the process

To summarize, this thesis contains the following main contributions:

1. Proposed a novel hybrid model called MOHA and a composed learning strategy for learning a hybrid automaton from continuous data.

2. Proposed a novel hybrid model called regression automaton and its inclined learning strategy for learning a hybrid automaton from continuous data.

3. Applied a probabilistic automaton learning approach for predicting an external agent's intention. A model predictive controller then uses such an intention to achieve a safe interactive control.

4. Developed a novel framework using timed automata for learning individual processes and Bayesian network for learning their dependencies. The model can be used as a diagnoser for anomaly detection.

5. Developed a translator called MO2SX filling the gap between MOHA and the state-of-the-art hybrid model checker SpaceEx for verifying the safety property of the data-driven MOHA model.

The above techniques deal with fundamental problems about learning, diagnosing, and verification of intelligent control systems. They have been implemented, evaluated, and applied to several case studies to demonstrate effectiveness and applicability in practice.

SAMENVATTING

Automatische regeltechniek is een techniek om regelaars te ontwerpen voor het besturen van machinale processen zonder menselijke tussenkomst. Het ontwerpen van regelaars met behulp van conventionele regeltechniek vereist echter een ontwerp op basis van eerste beginselen die volgen uit een volledig begrip van de omgeving en het systeem, wat onhaalbaar is voor complexe regeltaken zoals het rijden in een zeer onzekere verkeerssituatie. Intelligente regeltechniek biedt nieuwe mogelijkheden om de regelaarsstrategieën van mensen af te leiden door ons regelgedrag na te bootsen door middel van menselijke voorbeelden. In dit proefschrift richten we ons op intelligente regeltechnieken vanuit twee aspecten: (1) hoe men de regelaarsstrategieën kan leren van begeleiders door gebruik te maken van de beschikbare voorbeelddata; (2) hoe men kan verifiëren of de van data geleerde regelaar het proces veilig zal regelen.

Samenvattend bevat dit proefschrift de volgende hoofdbijdragen:

1. Een nieuw hybride model genaamd MOHA en een samengestelde leerstrategie voor het leren van een hybride automaat van continue data zijn voorgesteld.

2. Een nieuw hybride model genaamd “regression automaton" (regressie-automaat) en de geneigde leerstrategie voor het leren van een hybride automaat van continue data zijn voorgesteld.

3. Een benadering is toegepast voor het leren van probabilistische automaten om de intentie van een externe agent te voorspellen. Een modelvoorspellende regelaar gebruikt vervolgens een dergelijke intentie om een veilige interactieve regeling te bereiken.

4. Een nieuw raamwerk is ontwikkeld met behulp van getimede automaten voor het leren van individuele processen en Bayesiaanse netwerken voor het leren van hun afhankelijkheden. Het model kan worden gebruikt als een diagnose voor anomaliedetectie.

5. Een omzetter genaamd MO2SC is ontwikkeld. Deze vult het gat op tussen MOHA en de nieuwe hybride modelcontroleur SpaceEx voor het verifiëren van de veiligheidseigenschappen van het datagestuurde MOHA-model.

Bovenstaande technieken gaan om met fundamentele problemen over het leren, diagnosticeren en verifiëren van intelligente regelsystemen. Ze zijn geïmplementeerd, geëvalueerd en toegepast op verschillende gevallen uit de praktijk om de effectiviteit en toepasbaarheid in de praktijk aan te tonen.

Curriculum Vitæ

Qin was born on 4th December 1988 in Foochow, China. He graduated from Lianjiang No. 1 Middle School in 2007, where his interests in Chinese literature, mathematics, and physics grew quickly.

He obtained his bachelor's and master's degree both in automatic control. His academic research started in 2011 when he joined Prof. Jun Wang's lab at Tongji University, Shanghai. He found the joy of doing research, writing and reading papers at Tongji, where he also got fruitful research outcomes in signal processing and data mining with his professor. This experience gently opened a door for him to explore a much wider world. More importantly, it has planted the seed in his mind to devote himself to science and engineering and become a scholar.

It was by chance that he got a Ph.D. researcher position offered by Dr. Sicco Verwer at TU Delft. Sicco is a young researcher with rich experience in machine learning, especially in automata learning. Qin finally joined the exciting tide of machine learning. Because of his previous background in automatic control and signal processing, he is enthusiastic in applying automata learning in control and time series mining problems. He self-studied other related topics such as model checking. All of these dramatically broaden his horizon. He connected a link with his old colleagues at Tongji and several international collaborators who are also interested in automata-related theory. He was encouraged by them to apply his knowledge into application domains such as autonomous driving and cyber-physical systems' security. He also assisted his professor with several graduate-level courses and writing research proposals.

Qin is going to continue his research on safety verification for machine learning-enabled components of autonomous vehicles under the supervision of Prof. John M. Dolan at Carnegie Mellon University.

Publication List

- Journal articles (during Ph.D. study)

1. **Qin Lin**, Yihuan Zhang, Sicco Verwer, and Jun Wang. MOHA: a Multi-mode Hybrid Automaton Model for Learning Car-following Behaviors. IEEE Transactions on Intelligent Transportation Systems, 20(2): 790–796, 2019
2. Yihuan Zhang, **Qin Lin**, Jun Wang, Sicco Verwer, and John Dolan. Lane-change Intention Estimation for Car-following Control in Autonomous Driving. IEEE Transactions on Intelligent Vehicles, 3(3): 276–286, 2018

- Refereed workshop and conference papers (during Ph.D. study)

3. **Qin Lin**, Christian Hammerschmidt, Gaetano Pellegrino, and Sicco Verwer. Short-term Time Series Forecasting with Regression Automata. In ACM

SIGKDD 2016 Workshop on Mining and Learning from Time Series (MiLeTS), 2016

4. Gaetano Pellegrino, **Qin Lin**, Christian Hammerschmidt, and Sicco Verwer. Learning Behavioral Fingerprints from Netflows Using Timed Automata. In Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on, pages 308–316. IEEE, 2017
5. Christian Hammerschmidt, Sicco Verwer, **Qin, Lin**, and Radu State. Interpreting Finite Automata for Sequential data. In Interpretable ML for Complex Systems NIPS 2016 Workshop, 2016
6. Gaetano Pellegrino, Christian Hammerschmidt, **Qin Lin**, and Sicco Verwer. Learning Deterministic Finite Automata from Infinite Alphabets. In International Conference on Grammatical Inference, pages 120–131, 2017
7. Xiaoran Liu, **Qin Lin**, Sicco Verwer, and Dmitri Jarnikov. Anomaly Detection in a Digital Video Broadcasting System Using Timed Automata. In Thirty-Second Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) Workshop on Learning and Automata (LearnAut), 2017
8. Yihuan Zhang, **Qin Lin**, Jun Wang, and Sicco Verwer. Car-following Behavior Model Learning Using Timed Automata. IFAC-PapersOnLine, 50(1): 2353–2358, 2017
9. Yihuan Zhang, Jun Wang, **Qin Lin**, Sicco Verwer, and John Dolan. A Data-driven Behavior Generation Algorithm in Car-following Scenarios. In Dynamics of Vehicles on Roads and Tracks Vol 1: Proceedings of the 25th International Symposium on Dynamics of Vehicles on Roads and Tracks (IAVSD 2017), page 227-232. CRC Press, 2017
10. **Qin Lin**, Sridha Adepu, Sicco Verwer, and Aditya Mathur. TABOR: Agraphical Model-based Approach for Anomaly Detection in Industrial Control Systems. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pages 525–536. ACM, 2018
11. **Qin Lin** and Sicco Verwer. Learning a Provably Safe Adaptive Cruise Controller from Human Driving Data (submitted)
12. **Qin Lin**, Sicco Verwer, Robert Kooij and Aditya Mathur. Using Data Sets from Industrial Control Systems for Cyber Security Education, 14th International Conference on Critical Information Infrastructures Security, CRITIS 2019, Linköping, Sweden, 2019
13. **Qin Lin** and Sicco Verwer. Probabilistic Model Learning from Noisy Data. In International Conference on Grammatical Inference, pages, 2017 (extended abstract)

- Publications before Ph.D. study

14. **Qin Lin** and Jun Wang. Vertically Correlated Echelon Model for the Interpolation of Missing Wind Speed Data. IEEE Transactions on Sustainable Energy, 5(3): 804–812, 2014

15. Huajie Gu, Jun Wang, **Qin Lin**, and Qi Gong. Automatic Contour-based Road Network Design for Optimized Wind Farm Micrositing. IEEE Transactions on Sustainable Energy, 6(1): 281–289, 2015
16. **Qin Lin**, Jun Wang, and Weiting Qiao. Denoising of Wind Speed Data by Wavelet Thresholding. In Chinese Automation Congress (CAC), 2013, pages 518–521. IEEE, 2013

SIKS DISSERTATION SERIES

2011 01 Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models
02 Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language
03 Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems
04 Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference
05 Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
06 Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage
07 Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction
08 Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues
09 Tim de Jong (OU), Contextualised Mobile Media for Learning
10 Bart Bogaert (UvT), Cloud Content Contention
11 Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective
12 Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining
13 Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling
14 Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets
15 Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval
16 Maarten Schadd (UM), Selective Search in Games of Different Complexity
17 Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness
18 Mark Ponsen (UM), Strategic Decision-Making in complex games
19 Ellen Rusman (OU), The Mind's Eye on Personal Profiles
20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach
21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems
22 Junte Zhang (UVA), System Evaluation of Archival Description and Access
23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media
24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics
26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns
28 Rianne Kaptein (UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure
29 Faisal Kamiran (TUE), Discrimination-aware Classification
30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions
31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science
33 Tom van der Weide (UU), Arguing to Motivate Decisions
34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training
36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach
37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization
39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games
40 Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development
41 Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control
42 Michal Sindlar (UU), Explaining Behavior through Mental State Attribution
43 Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge
44 Boris Reuderink (UT), Robust Brain-Computer Interfaces
45 Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection
46 Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
47 Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression
48 Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
49 Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

---

2012 01 Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda
02 Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

03 Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories
04 Jurriaan Souer (UU), Development of Content Management System-based Web Applications
05 Marijn Plomp (UU), Maturing Interorganisational Information Systems
06 Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks
07 Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
08 Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories
09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms
10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment
11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems
13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
14 Evgeny Knutov (TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems
15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.
16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance
18 Eltjo Poort (VU), Improving Solution Architecting Practices
19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution
20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval
22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction
24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval
25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
26 Emile de Maat (UVA), Making Sense of Legal Text
27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games
28 Nancy Pascall (UvT), Engendering Technology Empowering Women
29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval

30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making
31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning
33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON)
34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications
35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics
36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes
37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation
38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms
39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks
40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia
41 Sebastian Kelle (OU), Game Design Patterns for Learning
42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning
43 Withdrawn
44 Anna Tordai (VU), On Combining Alignment Techniques
45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions
46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior
48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data
49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions
50 Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering
51 Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

---

2013 01 Viorel Milea (EUR), News Analytics for Financial Decision Support
02 Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
03 Szymon Klarman (VU), Reasoning with Contexts in Description Logics
04 Chetan Yadati (TUD), Coordinating autonomous planning and scheduling
05 Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns
06 Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
07 Giel van Lankveld (UvT), Quantifying Individual Player Differences
08 Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

09 Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications
10 Jeewanie Jayasinghe Arachchige (UvT), A Unified Modeling Framework for Service Design.
11 Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services
12 Marian Razavian (VU), Knowledge-driven Migration to Services
13 Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
14 Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning
15 Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications
16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation
17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid
18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification
19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling
20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval
21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation
22 Tom Claassen (RUN), Causal Discovery and Logic
23 Patricio de Alencar Silva (UvT), Value Activity Monitoring
24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning
25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System
26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning
27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance
28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience
29 Iwan de Kok (UT), Listening Heads
30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support
31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications
32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development
33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere
34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search
35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction
36 Than Lam Hoang (TUe), Pattern Mining in Data Streams

37 Dirk Börner (OUN), Ambient Learning Displays
38 Eelco den Heijer (VU), Autonomous Evolutionary Art
39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems
40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games
41 Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
42 Léon Planken (TUD), Algorithms for Simple Temporal Reasoning
43 Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts

---

2014 01 Nicola Barile (UU), Studies in Learning Monotone Models from Data
02 Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method
03 Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions
04 Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
05 Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability
06 Damian Tamburri (VU), Supporting Networked Software Development
07 Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior
08 Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints
09 Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
10 Ivan Salvador Razo Zapata (VU), Service Value Networks
11 Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support
12 Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control
13 Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains
14 Yangyang Shi (TUD), Language Models With Meta-information
15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare
16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria
17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability
18 Mattijs Ghijsen (UVA), Methods and Models for the Design and Study of Dynamic Agent Organizations
19 Vinicius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support
20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link
21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments

22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training
23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era
24 Davide Ceolin (VU), Trusting Semi-structured Web Data
25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction
26 Tim Baarslag (TUD), What to Bid and When to Stop
27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty
28 Anna Chmielowiec (VU), Decentralized k-Clique Matching
29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software
30 Peter de Cock (UvT), Anticipating Criminal Behaviour
31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support
32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data
33 Tesfa Tegegne (RUN), Service Discovery in eHealth
34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.
35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach
36 Joos Buijs (TUE), Flexible Evolutionary Algorithms for Mining Structured Process Models
37 Maral Dadvar (UT), Experts and Machines United Against Cyberbullying
38 Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.
39 Jasmina Maric (UvT), Web Communities, Immigration, and Social Capital
40 Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education
41 Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text
42 Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models
43 Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments
44 Paulien Meesters (UvT), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.
45 Birgit Schmitz (OUN), Mobile Games for Learning: A Pattern-Based Approach
46 Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity
47 Shangsong Liang (UVA), Fusion and Diversification in Information Retrieval

---

2015 01 Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response
02 Faiza Bukhsh (UvT), Smart auditing: Innovative Compliance Checking in Customs Controls
03 Twan van Laarhoven (RUN), Machine learning for network data
04 Howard Spoelstra (OUN), Collaborations in Open Learning Environments
05 Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding

06 Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
07 Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis
08 Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
09 Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems
10 Henry Hermans (OUN), OpenU: design of an integrated system to support lifelong learning
11 Yongming Luo (TUE), Designing algorithms for big graph datasets: A study of computing bisimulation and joins
12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
13 Giuseppe Procaccianti (VU), Energy-Efficient Software
14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations
15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation
16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork
17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
19 Bernardo Tabuenca (OUN), Ubiquitous Technology for Lifelong Learners
20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination
21 Sibren Fetter (OUN), Using Peer-Support to Expand and Stabilize Online Learning
22 Zhemin Zhu (UT), Co-occurrence Rate Networks
23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage
24 Richard Berendsen (UVA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation
25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection
26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure
27 Sándor Héman (CWI), Updating compressed colomn stores
28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO
29 Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
30 Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning
31 Yakup Koç (TUD), On the robustness of Power Grids
32 Jerome Gard (UL), Corporate Venture Management in SMEs
33 Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources
34 Victor de Graaf (UT), Gesocial Recommender Systems
35 Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

2016 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines

02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UVA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UVA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (UvT), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring

32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

---

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UVA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VU) , Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior

11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TUE), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UVA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VU), Logics for causal inference under uncertainty
23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VU), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (UvT), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VU), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems

41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TUE), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
11 Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks
12 Xixi Lu (TUE), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TUE), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araujo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis

24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
27 Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (UVT), Emotion Recognition from Mandarin Speech
30 Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

---

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
03 Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
05 Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
06 Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
09 Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VU), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection

22 Martin van den Berg (VU),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification