Fakultät für Informatik, Informatik 26 - Data Analytics and Machine Learning

Specification Mining in High dimensional Heterogeneous Data Sets of Large-Scale Distributed Systems

Artur Mrowca

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigten Dissertation.

Vorsitzende/-r: Prof. Dr. Uwe Baumgarten

Prüfende/-r der Dissertation:

1. Prof. Dr. Stephan Günnemann
2. Prof. Dr. Sebastian Steinhorst

Die Dissertation wurde am 04.12.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 17.03.2021 angenommen.

Abstract

In an increasingly interconnected world, distributed systems power our everyday lives. This includes multiple areas, ranging from automobiles and IoT to the domestic life and medicine. However, with this a lot of trust is put in those systems and its functioning, while unexpected behavior might lead to failures that cause serious damages.
Therefore, verification of distributed systems is essential, which includes the communication between devices, as well as its hardware and software.
In this work, the focus is on the verification of software. An increasingly important way for verification of distributed systems is to use traces that are recorded from system executions, which are used to detect, identify or explain errors. However, with growing size of such distributed systems this is increasingly challenging, e.g. as massive amounts of data are recorded, systems evolve, and the system is developed by multiple domains. To ensure verification of such systems often model checking is used, which is able to verify manifold nominal behaviors that were specified by respective domain experts. However, at a larger scale the manual generation of specifications becomes in-feasible, as functional variety is increasing and human cognition is limited. Therefore automated methods for Specification Mining are required. This includes methods that allow to better understand the system, as well as methods to automatically generate specifications from recorded traces.
This work presents a novel Specification Mining approach, which is aimed to be applied on traces of distributed systems.
Existing approaches for this, do not consider multi functionality, heterogeneity of data or exploitation of dimensional information in combination. Further, often no domain knowledge is included and specifications are limited in length or applicable to perfect traces only.
This is overcome in this work by proposing an end-to-end Data Mining pipeline that allows to extract specifications from raw traces. This is achieved by systematically breaking down complexity in dimension and procedure, as well as by modeling the recorded data under uncertainty with an appropriate model.
This pipeline contains six stages which are enabling this. Each of those steps requires dedicated characteristics. Therefore, in this work per stage multiple approaches are compared, developed and extended, which are presented here. Lastly, the overall consistency of the proposed approach is validated in an extensive case study of the automotive industry.

Zusammenfassung

In einer zunehmend vernetzten Welt gewinnen verteilte Systeme in zahlreichen Gebieten unseres Lebens immer mehr an Bedeutung. Im Zuge dessen genießen derartiger Systeme ein hohes Maß an Vertrauen. Dennoch kann ihr unerwartetes Verhaltens in schwerwiegenden Konsequenzen resultieren. Auf Grund dessen kommt der Verifikation von verteilten Systemen eine hohe Bedeutung zu. Dies beinhaltet die Korrektheit von Kommunikationsverhalten zwischen einzelnen Geräten, sowie die Korrektheit der Hardware und Software solcher Geräte.
Diese Arbeit behandelt den Softwareaspekt der Verifikation. Ein kosteneffizienter Weg der dies ermöglicht liegt in der datenbasierten Verifikation von Tracedaten, die aus Systemausführungen resultieren. Diese werden u.a. verwendet um Fehler zu entdecken, zu identifizieren und zu erklären. Mit zunehmender Größe derartiger Systeme wird dies jedoch stetig komplexer. Dies liegt zum Beispiel an der immensen Größe der Daten, an der stetigen Evolution der Systeme, sowie an der Tatsache, dass eine Vielzahl an Domänen bei der Entwicklung beteiligt sind.
Um dies zu lösen wird Model Checking verwendet, da diese Technologie in der Lage ist eine große Bandbreite an nominalem Verhalten zu validieren. Hierzu werden manuell Spezifikationen definiert, die unterschiedlichen Domänen entstammen können. Durch die immense Komplexität verteilter Systeme wird jedoch eine manuelle Erstellung unverhältnismäßig schwierig. Dies liegt insbesondere an der Funktionsvielfalt und der Beschränktheit der menschlichen Kognition. Aus diesem Grund werden automatisierte Methoden hierfür entwickelt, die unter dem Begriff des Specification Mining in der Literatur zu finden sind. Derartige Verfahren erlauben es zum Einen automatisiert Spezifikationen zu generieren und zum Anderen Systemverhalten besser zu verstehen.
Diese Arbeit beschreibt ein neuartiges Verfahren dieser Kategorie, dass die Extraktion von Spezifikationen aus verteilten Systemen immenser Größe ermöglicht. Existierende Ansätze beinhalten nicht alle notwendigen Charakteristika hierfür. Z.B. wird Multifunktionalität der Systeme oder die Heterogenität der Daten nicht beachtet. Das präsentierte Verfahren löst diese Herausforderungen, da es eine Ende-zu-Ende Lösung bietet, die den Prozess von rohen Daten bis hin zu extrahierten Spezifikationen beschreibt. Insbesondere wird die Komplexität der Daten heruntergebrochen. Basierend auf der reduzierten Repräsentation werden probabilistische Modelle verwendet, um Spezifikationen zu extrahieren.
Die Methode beinhaltet sechs Schritte, die dedizierte Charakteristika erfordern, um die Herausforderungen solcher Systeme zu bewältigen. Deshalb, werden für jeden dieser Schritte eine Vielzahl an Verfahren entwickelt, erweitert und verglichen. Zuletzt, wird die Validität der präsentierten Methodik an Hand mehrerer Datensätze aus der Automobilindustrie bewiesen und bewertet.

Contents

List of Figures

List of Tables

Glossary

BIC – Bayesian Information Criterion
BN – Bayesian Network

CAVI – Coordinate Ascent Variational Inference
CB – Constraint-Based
CI – Conditional Independence
CPD – Conditional Probability Distribution
CTBN – Continuous Time Bayesian Network
CTL – Computational Tree Logic

DAG – Directed Acyclic Graph
DBN – Discrete Bayesian Network
DBSCAN – Density Based Spatial Clustering of Applications with Noise
DM – Data Mining
DMS – Dynamical Multifunctional System

ELBO – Evidence Lower Bound
EM – Expectation Maximizations

FSA – Finite State Automaton

GHC – Greedy Hill Climbing

HBN – Hybrid Bayesian Network

IoT – Internet of Things

JPD – Joint Probability Distribution

LTL – Linear Temporal Logic
LTS – Local Trace Segmentation

MAP – Maximum A Posteriori
MES – Multivariate Event Sequence
MLE – Maximum Likelihood Estimation
MMHC – Max-Min Hill Climbing Algorithm
MPE – Most Probable Explanation
MSS – Multivariate State Sequence

PGM – Probabilistic Graphical Model	
PM – Process Mining	
RCA – Root Cause Analysis	
RV – Random Variable	
SAX – Symbolic Aggregate approXimation	
SB – Score-Based	
SC – State Change	
SD – Structure Discovery	
SOM – Self Organizing Map	
TSCBN – Temporal State Change Bayesian Network	
TV – Temporal Variable	
VI – Variational Inference	

1 Introduction

Growing inter-connectivity of systems that are surrounding us is rapidly changing the way we live. In the morning, a virtual assistant informs us on the weather, our appointments at work and the traffic situation, while, at the same time, the car heater activates as it received a signal from our alarm clock that we woke up. Driving to work, the car reads our emotions providing the right music to play and the best route to take, while it informs other cars about hazards along the way.
The functionality that is underlying this comfort is enabled by computational devices that are part of bigger systems that are surrounding us. Such devices are communicating either as part of a common infrastructure, such as the Internet of Things (IoT), or within their own infrastructure, e.g. within automotive in-vehicle networks. Systems that are combined in that sense are referred to as *large-scale distributed systems* within this thesis. Here, large-scale refers to either a high number of executing processes (or nodes) or applications processing vast amounts of data [11].
As presented in the introductory example such systems are highly beneficial for us making it increasingly hard to refrain from their usage. However, our safety and even the working of our society, depends on the well-functioning of those systems. As a consequence, it needs to be ensured that those systems behave as expected in terms of their hardware, software and communication.

1.1 Diagnosis in Software of Large Scale Systems

In this thesis the focus is on the software aspect of verification of such systems. This aspect is getting increasingly important [2] due to several reasons.
First, sensors and processors are integrated in more and more products of different industries, including automotive, aerospace, medicine or the social sector as shown in Figure 1.1. Next, within complex internally-interconnected systems, such as cars or airplanes, the number of control units that are running distributed software is increasing. Safety and functionality are improved through redundancy and more powerful software and hardware architectures. In addition to that, running those systems requires a high degree of compatibility across functionality and communication behavior on both hardware and software level. Thus, growing amounts of interrelated components and functionalities aggravate their verification.
At the same time misbehavior of functions and execution failures in both individual systems and within their interconnected cooperation might have severe consequences. These can be outages or unexpected behavior, which cause high costs in terms of unplanned down times [12], customer dissatisfaction or even loss of life [13]. Therefore, a high effort needs to be put in the verification of such large-scale distributed systems.

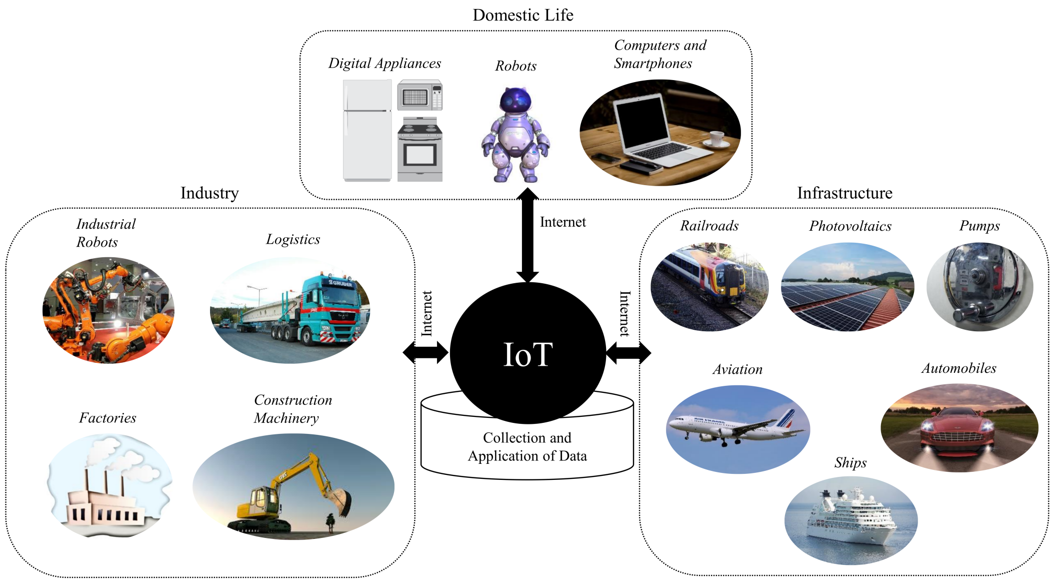


Figure 1.1: This image shows an overview of fields in which large-scale distributed systems are applied[1].

Software Verification: Two common ways for verification are fault injection and fault diagnosis. While in fault injection faults are specifically injected into the system in order to analyze resulting symptoms of the system, in diagnosis, faults are directly inferred from observed symptoms [14]. This work addresses approaches in the context of diagnosis. Diagnosis refers to tasks such as detection, localization, identification or prediction of known and unknown errors, as well as the determination of their root cause based on observations that are made from the system.

In diagnosis, multiple types of errors might be investigated. Among others, this includes scale dependent, configurational, behavioral or communicational errors. For instance, in automotive, bus overload might lead to functions remaining inactive, disabled components hinder requested functionality and unexpected driver behavior needs to be handled. Such errors are not always visible, e.g. due to the faults' characteristics, fault tolerance mechanisms that are built into the system or the lack of monitoring functionality in the system [14], making diagnosis of such errors challenging. This effect is amplified with the scale of the system, i.e. growing size yields more complex distributed systems with more functions, sensors and actuators. The consequence of this is an increasing amount of possible fault types with more and more complex patterns.

In particular, errors in software of large-scale distributed systems can manifest both within components or across components [11]. This makes verification challenging, as components are developed by distinct teams before being integrated to an overall system.

Therefore, during development of a large-scale distributed system or its participants, diagnosis is performed both before and after integration. The latter is particularly challenging, as a high degree of expertise across domains of the integrating parties is required, error patterns are more complex and multiple combinations of systems might be given.

[1] *Source:* `https://www.nidec.com/en-NA/product/new_field/iot/~/media/nidec-com/technology/new_field/iot/img/img_iot_01e.jpg`

1.2 Data-Driven Verification

An increasingly important way to verify software behavior of integrated systems systematically is by recording and verifying traces from system executions. Using traces is beneficial, as it is both cost-efficient and well-suited to systematize.
This type of verification can be done in multiple ways. First, those traces can be analyzed manually for misbehavior in an exploratory manner based on an expert's knowledge and his experience [15]. However, this requires a lot of effort and is often only possible for known errors.
Second, another common approach to verify correctness of integrated systems is by using model checking, where nominal behavior is defined and used to verify observed behavior. Definition of nominal behavior is done in a comprehensible manner using a set of rules. Each rule (also referred to as specification) can be defined, e.g. by using model checking languages which specify states the system is allowed to transition to. Those specifications are ran on observed traces or systems. If the observed behavior violates any specification, the respective erroneous part of the trace or system was identified. This allows to check for specific behavior of a set of functionalities systematically on large sets of traces. Especially for software of increased scale this is important, as manual localization of errors becomes in-feasible and such methods are well able to guide developers to the locations of faults with minimal human intervention [16]. A problem with this type of approaches is that currently such rules are mostly defined manually. However, with growing complexity of systems this becomes intractable as there are vast numbers of possible execution branches, generation is time consuming, multiple domains are involved in the process, high expertise is required and human cognition is limited.
Third, automated methods that operate directly on the data to diagnose known and unknown errors are becoming increasingly important. A growing branch of such methods are those that apply Machine Learning to infer knowledge for verification from recorded data. On the one hand, this is done with black box models, such as Neural Networks. Such approaches are helpful to use for tasks such as anomaly detection, error prediction or classification of unspecific errors [16, 12, 17]. On the other hand, such knowledge inference is done on the basis of white box models (e.g. using Bayesian Networks). This type of approaches is preferable to use in tasks that require interpretable results, e.g. during testing of functionalities.

Specification Mining: A field of research that combines both model checking and data-driven learning approaches is referred to as Specification Mining. Instead of manually defining nominal behavior as a set of rules (= specifications), such as in classical model checking, in Specification Mining automated approaches are developed to learn such rules from software code or execution traces. For instance, Machine Learning is used to learn and model observed behavior and inference is used to identify rules that most likely represent correct system behavior. Using such techniques allows to generate a more complete set of rules, while manual effort is reduced.
Current approaches in the field of Specification Mining are not designed for large-scale distributed systems. For instance, those methods do not assume combinations of multiple functional behaviors with high-dimensional, massive and heterogeneous data sets. However, traces are expected to grow in size and complexity, making it essential to solve

this task for such massive data sets. As described in [2], there will be nearly 26 billion devices on the Internet of Things by 2020 and, as indicated in [4], in-vehicle network complexity increased to up to 90 interacting components with traces that have grown to massive amounts. In [16] this task was described as highly complex by stating that "although the complete execution trace of a program is a valuable resource [...], the huge volume of data makes it unwieldy for usage in practice".
A promising direction to solve this is the usage of Specification Mining at a large scale. Doing this becomes possible with the current advent of Big Data technology, such as Apache Hadoop [18], in combination with Machine Learning. Less recent work dealt with this topic, while applying this at scale might allow to perform systematic verification of future distributed systems to ensure its safety and quality.
Therefore, in this work, those methods are combined to a systematic extraction approach which addresses the above issues. The proposed method is a Data Mining (DM) pipeline that systematically reduces, prepares and structures recorded traces. From the resulting representation, the expert guides the extraction of dominant system states, that, first, can be used as templates for specifications and, second, improve the expertise of the expert and thus, supports the experts ability to generate specifications. The same representation is further used in this pipeline to semi-automatically extract specifications described in Linear Temporal Logic (LTL).

Challenges: Mining of specifications in large-scale distributed systems has multiple challenges.
First, this includes the type of data that needs to be dealt with. Recorded traces are often massive and in raw data format with a high degree of redundancy in the data. Therefore, it needs to be interpreted and reduced meaningfully. Above that, this data is temporal, high dimensional and heterogeneous. For an automated procedure those data types need to be homogenized and further reduced in dimension.
Second, recorded data represents multiple functional behaviors. Different dimensions and segments in the analyzed trace correspond to particular functional groups that are relevant to experts of particular domains only. Such functional groups need to be found (semi-) automatically. Especially, in massive traces, this requires to break down complexity by identifying functional aspects.
Third, large systems are designed by multiple experts from different domains. As a consequence, testing of those systems is aggravated as individual functional behaviors cover multiple fields of knowledge, expertise is distributed, cross-functional system knowledge is represented implicitly (i.e. experience) and hidden correlations of subsystems exist. To overcome this, the lack of domain knowledge needs to be covered by guiding the inspecting experts towards relevant functional procedures and dimensions for diagnosis. This allows for identification of relevant functional entities and consequently, for extraction meaningful specifications.
Fourth, when given a set of observed temporal behaviors within a functional group, a consequent step is to perform specification extraction. Challenges of this extraction include handling of imperfect traces and finding specifications of arbitrary length. This is solved by appropriately representing traces, which is commonly done by using either rule-sets or models. For the case of imperfect traces, probabilistic models are promising to handle such data types for the given scenario. In case of the given type of system,

this requires to represent multivariate temporal sequences in a way that dimensional information is preserved while noise is handled.
Fifth, granularity of found functional groups is often subjective and depends on the analyzing domain, e.g. it can be on a communication layer where messages are considered events or on a system state layer, where changes in system state are analyzed. Thus, again domain knowledge needs to be included in the process of specification mining.

1.3 Contributions and Research Questions

1.3.1 Contributions

This work deals with the task of extraction of specification as well as the determination of dominant procedures in traces of large-scale distributed systems. In particular this includes the following contributions.

- **Design of a Data Mining Pipeline:** Based on the shortcomings that were identified for the existing approaches for specification mining and dominant behavior extraction, a novel approach is presented to overcome those. It allows to extract specifications per domain through a semi-automated procedure that involves expert input. By extracting functional groups and by reducing data meaningfully, complexity of the inspected large scale system is broken down and traces are analyzed in terms of aspects relevant to the domain. Further, this approach is modular which allows for variation of individual stages within this pipeline. Lastly, an end-to-end solution is presented, that covers all stages from the raw trace to sets of relevant specifications.

- **Pipeline Stages:** For each stage of the approach, multiple variations can be used, which in general depend on the data used. In this work, each of those is discussed and evaluated in order to allow for a good choice of methods per stage.

- **Extensive Evaluation:** To demonstrate its validity and effectiveness, the proposed approach was evaluated on multiple data sets from the automotive industry. There, all stages are evaluated in detail. This includes validation of each step, discussion of hyper parameters in dependence of the data set and the discussion of conclusions that are drawn from this. The evaluation shows that the proposed DM pipeline allows for Specification Mining at large scale.

1.3.2 Research Questions

Within the above contributions multiple research questions arise that need to be answered within the design of the DM pipeline. Those are the following.

1. How does a systematic Specification Mining approach need to be designed to be integrable in current testing and verification procedures of large-scale distributed systems?

2. How can temporal structure be exploited to allow for a more expressive Specification Mining on MSSs?

3. How can Specification Mining be performed on noisy and heterogeneous traces in order to produce specifications that compare multiple data types?
4. How can domain-specific Specification Mining be performed and experts included in the mining procedure?
5. How can the complexity of a multi-functional large-scale distributed system be broken down, such that effective and efficient mining of relevant specifications is enabled?
6. How can behavior of functional procedures of MSSs be represented under uncertainty and specifications of arbitrary length extracted?
7. How can raw communication traces of large-scale distributed systems be processed and functional procedures, as well as specifications identified from those?
8. How can functional procedures be identified in high-dimensional MSSs of large-scale?
9. Which combination of approaches is suited to be used at each individual step of the semi-automated processing pipeline?

After presenting the pipeline in the further course of this thesis, in Chapter 10 answers to the above questions will be given based on the conclusions drawn from the individual steps of the pipeline.

1.4 Thesis Outline

The focus of this thesis is on Specification Mining for the identification of functional errors from traces of large-scale distributed systems. Such traces needs to be represented in an appropriate data format. Also, an understanding of the formal connection between the system, its state over time, the functional perspective and the recorded trace is required. Those preliminary aspects are introduced in Chapter 2. Based on those assumptions in Chapter 3 the novel DM pipeline for Specification Mining is presented and put into context of related work in the field. In the successive chapters the individual steps of this pipeline are presented and approaches for those steps are evaluated for suitability in the pipeline. An automated preprocessing approach for communication traces is described in Chapter 4, a clustering approach for multivariate temporal data in Chapter 5, a segmentation clustering approach to identify correlating MSSs in Chapter 6, a model for representation of MSSs under uncertainty in Chapter 7 and two inference approaches to identify specifications from this model in Chapter 8. Based on the results that are presented in those chapters an implementation of the DM pipeline is presented and evaluated on multiple data sets of the automotive industry in Chapter 9. Lastly, a conclusion is given, which summarizes the results of this work in Chapter 10

1.5 List of Publications

The following publications resulted from research performed during the time of this doctoral thesis, while only publications [1-4] are referenced in this work.

1. Artur Mrowca, Florian Gyrock, Stephan Günnemann. Temporal State Change Bayesian Networks: Modeling Multivariate State Sequences with evolving dependencies. *Under Review.*

2. Artur Mrowca, Martin Nocker, Sebastian Steinhorst, Stephan Günnemann. Learning Temporal Specifications from Imperfect Traces Using Bayesian Inference. *Proceedings of the 56th Design Automation Conference (DAC 2019).*

3. Artur Mrowca, Barbara Moser, Stephan Günnemann. Discovering Groups of Signals in In-Vehicle Network Traces for Redundancy Detection and Functional Grouping. *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD 2018).*

4. Artur Mrowca, Thomas Pramsohler, Sebastian Steinhorst, Uwe Baumgarten. Automated Interpretation and Reduction of In-Vehicle Network Traces at a Large Scale. *Proceedings of the 55th Design Automation Conference (DAC 2018).*

5. Jan-Philipp Schulze, Artur Mrowca, Elizabeth Ren, Hans-Andrea Loeliger, Konstantin Böttinger. Context by Proxy: Identifying Contextual Anomalies Using an Output Proxy. *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD 2019).*

6. Peter Wolf, Artur Mrowca, Tam Nguyen, Bernard Bäker, Stephan Günnemann. Pre-ignition Detection Using Deep Neural Networks: A Step towards Intelligent Automotive Diagnostics. *Proceedings of the Intelligent Transportation Systems Conference 2018 (ITSC 2018).*

7. Tam Nguyen, Artur Mrowca, Barbara Moser, Andreas Jossen. Analysing the load on electric vehicles using unsupervised segmentation models as enabler to determine the time of battery replacement and assess driving mileage. *Proceedings of the 13th Conference on Ecological Vehicles and Renewable Energies (EVER 2018).*

2 Preliminaries

First, multiple terms are introduced, which are used in the context of fault diagnosis in the verification community. Second, the proposed approaches are aimed to analyze large-scale distributed system. Therefore, those systems are introduced on a systematical and a functional level together with the type of data that those produce. Lastly, in this work the focus is on verifying behavioral errors in software of distributed systems, which is why those are introduced in the last part of this chapter.

2.1 Terminology

2.1.1 Verification

Distributed System: According to [1] a distributed system is a set of components that are located on different nodes, which coordinately perform an action by exchanging communication messages. These components can be seen as a collection of computational devices that appear to its users as a single coherent system [1].

Diagnosis: Diagnosis is the identification of the nature of a problem by examination of the observed symptoms [14]. Those are tasks such as fault localization, root cause identification, fault prediction and fault detection.

Event: Events are exceptional conditions that occur when running the system. In the scope of this thesis this includes all conditions of the system, e.g. its state changes.

Failure: A failure is a system state in which a service or an application deviates from its correct behavior. It is an error that is observable from outside the system [16].

Error: An error is a certain condition within the system that may have led to a failure. It is caused by one or more faults and is a discrepancy between a condition of the system and its theoretically correct condition [16].

Fault (= bug): A fault is defined as the underlying cause of an error. Faults are events that can cause other events but are not caused by other events. Faults can be permanent if the fault persists until reparation, intermittent if those are discontinuous and transient if those are temporary. [16].

Symptom: A symptom is a subset of observed system states that indicates a system's misbehavior. It is an external manifestation of failures and might be visible indicators that a failure happened (e.g. anomaly) or the direct observation of a failure [14].

Failure Mode: The failure mode is a possible way that a system can fail. A complex system has multiple failure modes [16].

Root Cause: The root cause of a failure or of a fault obtains the symptoms that led to the failure or fault [2].

Explanation: The explanation of a failure or fault explains how the root cause is linked to the symptoms [2].

Bohrbug: A software bug which manifests reliably under a well-defined, but possibly unknown set of conditions [16], as opposed to **Mandelbugs**. Mandelbugs are complex and unpredictable. Those include performance bugs, memory leaks, software bloats and security vulnerabilities [16].

2.1.2 Data Types

Temporal Variable (TV): A temporal variable S refers to a dimension of information that is transmitted, with a value space of $\hat{V}$. Each TV S might be of numerical, binary, ordinal or nominal type with events or states with value $\hat{v} \in \hat{V}$. That is for the numerical case $\hat{v} \in \mathbb{R}$, binary case $\hat{v} \in \{0, 1\}$, ordinal case $\hat{v} \in \mathbb{N}$, nominal case $\hat{v} \in \bar{S} = \{s_1, s_2, ...\}$.

Multivariate Event Sequence (MES): A MES $\bar{M}$ is defined as a series of events $\bar{E}_i = (t, S_k, s_j)$ that each represent a state s_j of a TV S_k at a defined time t. It is ordered according to its times of occurrence:

$$\bar{M} =< \bar{E}_1, \bar{E}_2, ... > \text{ such that } \bar{E}_i.t \leq \bar{E}_{i+1}.t \tag{2.1}$$

Multivariate State Sequence (MSS): Formally, a MSS M is a series of state intervals $E_i = (s, e, S_k, s_j)$, defined by start times s, end times e and states s_j they are in. It is ordered according to their start times and each assigned a TV S_k:

$$M =< E_1, ..., E_I > \text{ such that } E_i.s \leq E_{i+1}.s, E_i.e = E_{i+1}.s \tag{2.2}$$

In particular, in MSSs each TV is in a state at all times and only state changes are observed. Moreover, MESs record states of TVs at defined points in time. In contrast to that, MSSs store intervals of states that TVs are in. In this context, each MES can be transformed to a MSS by filtering MESs such that two consecutive sequence elements of the same TV, that have same value, are filtered out.

2.2 System Definition

2.2.1 Distributed Systems

The DM pipeline that is proposed in this thesis is aimed to support trace diagnosis of large-scale distributed systems. Here, the term large scale refers to a system which has a large number of applications running and consequently produces vast amounts of data (as opposed to ingesting big amounts of data [11]). Such systems can be formalized

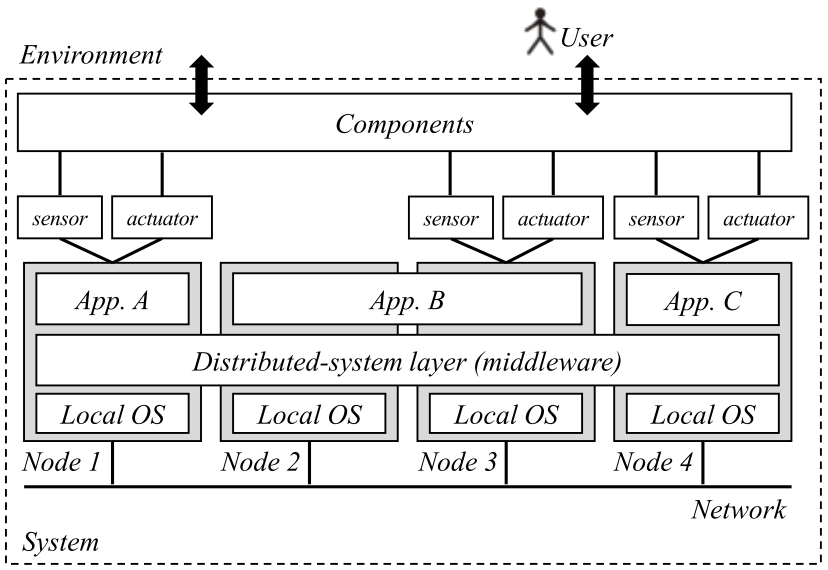


Figure 2.1: A formalization of a distributed system [1] is shown.

as shown in Figure 2.1, which was taken from [1], and are introduced in detail in the following.
A distributed system is surrounded by an environment and one or more users that are interacting with it. Within the system, multiple computing nodes run distributed applications that might be executed on one or multiple nodes. Moreover, multiple applications may run on one node. Each node is connected and interacting on a common network and interacts with hardware components by receiving sensory input from those or by transmitting actuatory input to those. Above that, components are connected to the environment and to the users.
The goal of the proposed diagnosis is on learning behavioral patterns of such systems in order to use the gained knowledge for verification of functional correctness. This is done by inspecting data recorded from traces of system executions. It is assumed that the system records the state of the environment, the state of the user interaction, as well as calls of the applications that describe functional operations and the state of the system. For example, in the distributed software system that runs in a car, the system state might be the speed of the car or the state of the left front door. Each application that is running on the system nodes is modifying the state of sub-components as well as other nodes of the system, where both nodes and components are coupled.
Further, unlike in component tests, where the goal is to verify that individual parts of the system behave as expected, here the focus is on integration tests. In those types of tests the goal is to ensure that the overall integrated system behaves as expected. This correctness can be verified using data-driven verification methods. For this, traces are recorded which contain the states of the users, the environment and the overall system. Those traces are analyzed for diagnosis of errors and failures.

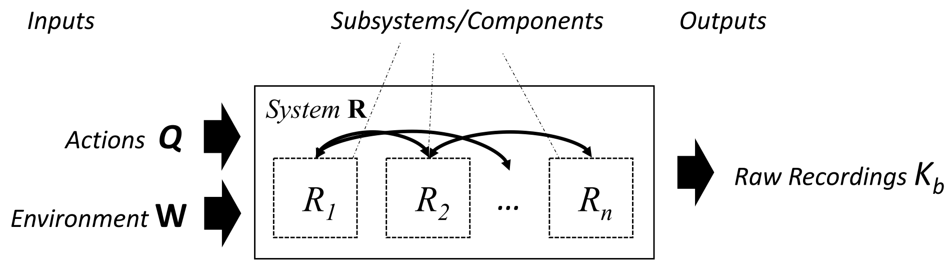


Figure 2.2: The simplified representation of a large-scale distributed system is shown, which is referred to as DMS in the scope of this thesis.

2.2.2 Simplified System State Perspective

In the following, a black-box system formalization is deduced from the above definition. This representation will allow to formalize and introduce the proposed DM pipeline, as well as to define the scope of applicability of it. In particular, this includes the linkage between the system states, its functions and its executions, which are presented here.

System representation: The simplified representation, which is called a Dynamical Multifunctional System (DMS) in the scope of this work, is shown in Figure 2.2. There, the system state $\mathbf{R}$ is abstracted into interconnected sub-component states $\mathbf{R}_i$. In terms of the above definition, sub-components $\mathbf{R}_i$ are either nodes or components, e.g. $\mathbf{R}_1$ could be the door and $\mathbf{R}_2$ the left rear wheel of a car. As specified above, the overall system state $\mathbf{R}$ depends on the previous system states, user actions and the environmental states. To verify the correctness of this system, it is executed under multiple test scenarios. Those produce a trace which contains multiple variants of system executions, which need to be analyzed using data-driven approaches for verification.

Dynamical Multifunctional System: In the scope of this work we define a DMS $\mathbf{R}$ as the state of an integrated system that consists of K interrelated subsystems with states $\mathbf{R}_k$ with $k \in [1...K]$, that may or may not interact. A DMS has a system state $\mathbf{R}(t)$ it is in at any point in time t and has a set of actions $\mathbf{Q}$ and a set of environment variables $\mathbf{W}$ as input. Thus, recorded behavior of those variables is represented in an trace $\mathbf{K}_b$ that is produced by the system.

Action: The set of actions $\mathbf{Q}$ defines the state of multiple temporal input variables, which are external actions that interact with the system state. This variable has a defined state $\mathbf{Q}(t)$ it is in at any time t.

Environment: The set $\mathbf{W}$ is the state of multiple TVs that define the environmental influence on the system. Those variables have a state $\mathbf{W}(t)$ those are in at any time t.

Total DMS State: The state $\mathbf{X}$ of the overall DMS is called the total state of the system. It is comprised of the system states $\mathbf{R}$, the actions $\mathbf{Q}$ and the environment variables $\mathbf{W}$, i.e. $\mathbf{X} = (\mathbf{Q}, \mathbf{W}, \mathbf{R})$. There, each dimension X_i of $\mathbf{X}$ is comprised of a TV

X_{tv} with a defined state X_{val}, i.e. $X_i = (X_{i,tv}, X_{i,val})$.

Dynamics: From a generative perspective it is assumed that each system state is produced from its preceding states. Starting from an initial state $\mathbf{R}^0$, the consecutive state $\mathbf{R}^{i+1}$ is produced by a mapping function f, that uses its previous actions $\mathbf{Q}^i$, environmental influences $\mathbf{W}^i$ and states $\mathbf{R}^i$ as input, i.e. $\mathbf{R}^{i+1} = f(\mathbf{Q}^i, \mathbf{W}^i, \mathbf{R}^i) = f(\mathbf{X}^i)$. Here, the iteration index indicates a particular point in time, in terms of a discrete time of the recording at that time of iteration.

From that perspective any trace $\mathbf{K}_b$ can be seen as the product of multiple iteration steps performed on $\mathbf{R}^{i+1} = f(\mathbf{X}^i)$, when starting from a defined $\mathbf{R}^0$.

2.2.3 Functional Perspective

2.2.3.1 Functions

A modern distributed system is highly complex, as applications run on multiple components and communicate on a common network, while components, applications and the network communication are developed by different groups of developers. In particular, this might result in bugs or hidden causalities.
Therefore, in data-driven integration tests the correct interplay of those components is verified by inspecting that its functional flow during run time is correct. For this, subsets of those components are tested. Further, it is assumed that if all functional flows of all functions are correct, the integrated black-box DMS behaves according to its nominal behavior and thus, is also correct.
The verification of such functional flows from recorded executions is the main intent of this thesis. This requires to introduce the notions of functions and the concept of functional procedures, which is done in the following.

Formal Definition: A function F_k of a DMS $\mathbf{R}$ is a defined group of processes of the DMS, that realize a closed subset of tasks that logically belong together. Each function F_k operates on a defined subset $\mathbf{R}_{sub}$ of subsystems $\mathbf{R}_k \in \mathbf{R}_{sub}$ and thus, modifies the system state of those subsystems. It depends on subsets of actions and of environment variables. Each function that operates on the system is denoted as $\mathbf{R}_k^{i+1} = F_k(\mathbf{Q}^i, \mathbf{W}^i, \mathbf{R}_k^i)$, which is a function that inputs the current state $\mathbf{R}_k^i$ of its subsystem and outputs its consequent state $\mathbf{R}_k^{i+1}$.

The totality of all functions F_k defines the procedural aspects of the overall DMS $\mathbf{R}$ and thus, the complete behavior that is present within the DMS. For the purpose of verification it, therefore, makes sense to define the system dynamics function $f(\mathbf{X}^i)$ that links consecutive system states, in terms of such functions. For this the totality of functions $\mathbf{F} = (F_1, F_2, F_3, ...F_n)$ is assumed, that can either be active or inactive at any iteration step, where the activeness of each function is defined by a vector $\mathbf{o}(\mathbf{X}) = (o_1(\mathbf{X}), o_2(\mathbf{X}), o_3(\mathbf{X}), ...o_n(\mathbf{X}))$, where $o_i(\mathbf{X})) = 0$ if the corresponding function is inactive and 1 other wise. With this $f(\mathbf{X}^i)$ can be written as

$$\mathbf{R}^{i+1} = f(\mathbf{X}^i) = \mathbf{o}^i(\mathbf{X}^i) \cdot \mathbf{F}^i(\mathbf{X}^i) \tag{2.3}$$

where $\mathbf{o}^i(\mathbf{X}^i)$ defines the set of active and inactive functions at iteration i.

2.2.3.2 Functional Procedures

In a DMS functions F_k are not modifying the system state randomly, but rather allow for a finite set of procedural patterns that modify the system state $\mathbf{R}$ (or rather its subsystems) within a defined time span in a defined manner. Such patterns, that a function may cause, are called functional procedures.

Formal Definition: Depending on the state $\mathbf{X}^i$, the start of the activity of function F_k (i.e. $o_k(\mathbf{X}^i)$ changes from 0 to 1) might trigger the execution of a defined procedure of system states $\mathbf{Q}$, $\mathbf{W}$ and $\mathbf{R}_{sub}$. The number of possible procedures of function F_k is finite and thus a set of functions f_{ki} is defined, that each produce a defined MES depending on $\mathbf{X}^i$. Those defined procedures are called functional procedures.

The set of all sub-functions of a function can be written as $F_k(\mathbf{Q}, \mathbf{W}, \mathbf{R}_{sub}) = (f_{k1}, ..., f_{km})$ and its activation at step i denoted as $\mathbf{o}_k(\mathbf{X}) = (o_{k1}(\mathbf{X}), ...o_{km}(\mathbf{X}))$. Here, $\mathbf{o}_k(\mathbf{X})$ is either a one-hot or a zero vector, as a functional procedure can be either currently active within a time span or not active. Thus, by inserting this into equation 2.3 the overall consecutive system state $\mathbf{R}^{i+1}$ could be written in terms of all functional procedures. In particular, any subsystem state $\mathbf{R}$ depends on all active functional procedures influencing it. Notably, the system state at iteration i depends on inputs that might have triggered functional executions, functional procedures of other active functions, actions and environmental conditions. Apart from this, the amount of interaction between functional procedures and other functions depends on the amount of overlapping subsystems $\mathbf{R}_{sub}$ that are influenced by the respective function. This idea of functional procedures is illustrated in Figure 2.3 for four functions F_k that each operate on different TVs of subsystems $\mathbf{R}_{sub}$ producing different functional procedures f_{ki}.

2.3 Analyzed Trace Data

2.3.1 Trace Recordings

The DMS under test is ran under multiple test scenarios and the execution recorded for verification. In the proposed DM pipeline this trace data is represented mainly in the three following formats.

Raw Recording: The raw recording $\mathbf{K}_b$ captures a sequence of one or more encoded total states $\mathbf{X}(t)$ at defined times t. Its elements are defined as tuples of shape $(t, enc(\mathbf{X}(t)))$, where enc is the encoding used for each state $\mathbf{X}(t)$.

Interpreted Recording: The interpreted recording $\mathbf{K}_s$ captures a MES that represents the total states $\mathbf{X}(t)$ at defined times t after being decoded and split according to their TV dimension, where the TV has an identifier s_{id} and a value v. Its elements are defined as tuples of shape $(t, X_i(t)) = (t, s_{id}(t), v(t))$ for components $X_i \in \mathbf{X}(t)$. Here,

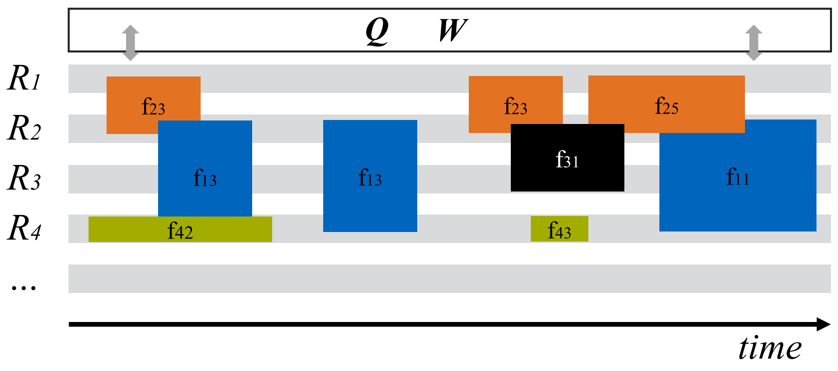


Figure 2.3: Multiple occurrences of functional procedures f_{ki} are shown that each are a part of a function F_k. Each F_k is represented with a different color here, e.g. f_{11} and f_{13} are functional procedures of F_1 (indicated in blue).

each dimension of $\mathbf{X}(t)$ refers to a TV.

Reduced Recording: The reduced recording $\mathbf{K}_n$ captures a MSS of the total states $\mathbf{X}(t)$, where each dimension contains the state a TV is in at any time t and only changes in state are observed. There, each element describes the start and end time of a state, as well as the corresponding TV and its state as tuple (s, e, s_{id}, v), where s are start and e end times of intervals of a TV defined by s_{id}. This representation is found by removing consecutive duplicates per TV in $\mathbf{K}_s$.

2.3.2 Functional Procedures and Traces

Functional procedures represent behavior of subsets of the system. These procedures are observed on a subset of TVs in a trace, and thus, can be similarly represented as either MESs or MSSs. Such procedures occur multiple times within the trace. Thus, it is assumed that the total observed trace is the composition of multiple functional procedures that are active or inactive across time and that might or might not interact with each other within certain time spans.
Each functional procedure produces a MES (called instance), where dimensions are TVs S_i of the system, which can be actions Q_i, environmental variables W_i or states of the system R_i. In case of a DMS, each TV can have a different data type which could be numerical, ordinal, binary or categorical. Apart from TVs with numerical data type all TVs S_i have a fixed set of states $\bar{S} = \{s_1, s_2, ...\}$ which a TV can be in at any point in time. For TVs with numerical data type the same can be achieved through quantization of its values. Further, in both the MES and MSS representation of a trace it is assumed that there are dynamically changing temporal-causal dependencies between multiple TVs S_k at many points in time.
This definition of functional procedures is useful for multiple tasks. Depending on the task any f_{ki} might capture procedures of different granularity. f_{ki} might capture an exact variant of a functional execution (e.g. 30 observations of $\langle$ press button, left light on $\rangle$) or it could capture multiple branches of the execution both in state and time (e.g.

10 observations of ⟨ press button, left light on ⟩, 20 observations of ⟨ press button, power down, left light off ⟩). The former, might be useful for specification mining, while the latter might be used for anomaly detection. Therefore, the right representation of f_{ki} depends on the mining task.

Data Characteristics: The data that is recorded as a raw trace is characterized as follows.

- *Raw trace:* The recorded data is raw in the sense that an encoded representation is initially stored. Thus, data needs to be preprocessed into MESs or MSSs.
- *Massive size:* The considered data is massive in size, e.g. in an in-vehicle network which resembles a DMS, this is in the range of about 10 million data points that are recorded per minute from the DMS.
- *High dimensional:* The number of TVs that define each state $\mathbf{X}$ in the DMS is high and potentially ranges from 1 000 to 10 000 dimensions.
- *Heterogeneous:* Multiple data types are present in the recordings which include nominal, ordinal, binary and numerical types.
- *Unstructured:* Data of multiple functions is recorded within one common trace. As the integrated system might contain hidden correlations data structure is not clear and thus, needs to be found.
- *Noisy and erroneous content:* Data recorded from DMSs might be noisy or erroneous, e.g. resulting from bad measurement devices.
- *Multifunctional:* Multiple functions from multiple domains are recorded from the integrated DMS. Those functions operate on common subsystems and thus, may interfere. In addition to that, functional procedures per function need to be identified.

Example: An example of a DMS is a car which consists of multiple subsystems such as the infotainment, the driving assistance or the light system, i.e. $\mathbf{R} = (\mathbf{R}_{inf}, \mathbf{R}_{da}, \mathbf{R}_{lig})$. In that system actions $\mathbf{Q}$ might be the activation of the blinker or the intervention of the driving assistance when pressing the braking pedal, i.e. $\mathbf{Q} = (\mathbf{Q}_{act}, \mathbf{Q}_{ped})^T$. Also environment variables, might be the outside brightness, which might influence if a driving assistance system is on or off $\mathbf{W} = (\mathbf{W}_{bright})^T$. The raw recording $\mathbf{K}_b$ might be occurrences of information captured throughout a 30 minute driving session, which could give $\mathbf{K}_b = \langle (t = 1s, \texttt{x5A x01}), (t = 2.7s, \texttt{x5C x01}), , (t = 4.2s, \text{x5C x02}), ... \rangle$, where `x5A` might encode the brightness recorded from the environment and `x01` might be the state of the left blinker, which might be on or off. The interpreted recording could thus, be $\mathbf{K}_s = \langle (t = 1s, \text{brightness}, 20\%), (t = 1s, \text{left blinker}, \text{on}), (t = 2.7s, \text{left blinker}, \text{on}), (t = 2.7s, \text{brightness}, 40\%), (t = 4.2s, \text{left blinker}, \text{off}), ... \rangle$. The resulting reduced representation could then read as $\mathbf{K}_n = \langle (t = 1s, \text{brightness}, 20\%), (t = 1s, \text{left blinker}, \text{on}), (t = 2.7s, \text{brightness}, 40\%), (t = 4.2s, \text{left blinker}, \text{off}), ... \rangle$.

2.3.3 Diagnosis and Fault Model

Goal of Diagnosis: The aim is to do diagnosis in complex distributed systems in a data-driven manner, i.e. identify problems in such systems by analyzing recorded traces. In particular this includes identifying and finding errors in a trace, as well as understanding errors of such systems to enable explanation of it. Failures are external manifestations of a system misbehavior and thus, are observed when those occur. Failures are the result of underlying errors. While not all errors result in failures, errors in a system might eventually lead to a failure. Therefore, it is crucial to locate errors, identify the faults which are the root cause for the errors, understand the correlations and explain the error. The focus of the investigation in this thesis is on finding and explaining errors (rather than failures or faults).

Relevant Types of Errors: In a distributed system, errors can be categorized into system-level or function-level errors. System-level errors include node-level errors, such as excessive workloads or memory leaking, as well as inter-node errors, such as communication errors. Function-level errors manifest in the behavior of the system in terms of its state $\mathbf{X}$. In integrated systems, verification of systems on a functional level allows to effectively analyze large-scale DMSs using data-driven testing methods. That is why, in the scope of this thesis, the focus is on this type of error.
In those types of errors, the correct condition is defined by valid states $\mathbf{X}$ the system can be in at any time. Here, static and dynamic (also called behavioral errors) function-level errors are distinguished. The former refers to errors that manifest in a single bad system state, e.g. a car might have the error state of an engine beings off and the rotational frequency of the engine being high. The latter refers to errors that manifest in a sequence of bad system states, e.g. a car might have the correct state of a sequence ⟨ button pressed, engine starts, engine running ⟩, while a possible error state would be ⟨ button pressed, engine off, engine not running ⟩. An extension to this are dynamic errors that include the system state and procedural information at the same time, e.g. the sequence ⟨ button pressed, engine starts, engine running ⟩ is valid if the keys are in the car, while an erroneous state would be that the keys are not in the car while the same sequence is observed. For both latter cases, in the context of the term functional procedures, two instances of the same functional procedure are observed. One with the correct behavior and another one with a bad behavior. Notably, this functional procedure includes steady system states, as described in the last case. Assuming all system states to be recorded in a trace, symptoms are invalid dynamic states of functional procedures that are present in the trace.
Multiple faults could have caused such errors, such as deadlocks in the code, hardware failure or simultaneous access to hardware resources by multiple threads.
In this thesis, the focus is on both dynamic errors as well as on the extension that includes the system state. This is done by building a model that is able to capture corresponding relationships both in terms of procedures and steady states, which will be presented in more detail in subsequent chapters. Additionally, it is assumed that faults of the distributed software are Bohrbugs, i.e. it is assumed that under identical conditions similar errors result.

Verification of Behavioral Errors: Functional procedures can have different instances. Those instances might correspond to correct or erroneous executions of the function, where erroneous instances exhibit behavioral errors. It is assumed that any data point in a trace corresponds to an instance f'_{ki} of a functional procedure f_{ki}. With this, the system is assumed to be correct (in terms of behavior), if no instance f'_{ki} contains behavioral errors. That is, if all observed executions of functions F_k are correct. If this is the case, all corresponding subsystems $\mathbf{R}_k$ in an observed trace of a DMS are correct in behavior. Thus, by considering each functional procedure separately for diagnosis, all error types and failure modes that are caused by behavioral errors can be verified. If correct functional procedures are known, those can be used to identify behavioral errors of corresponding observed instances of functional procedures using approaches such as model checking. This requires specifications that need to be generated per functional procedure. A procedure to do this for DMSs is presented in this work.

Dominant behavior: In order to identify behavioral errors in this work, multiple instances f'_{ki} of a functional procedure f_{ki} are observed. Those instances might vary in behavior, while sub-sequences of the procedures are identical. We refer to the sequence of sequence elements of f_{ki} that are shared most often among instances of f'_{ki} as dominant behavior.

3 Data Mining Pipeline for Systematic Diagnosis of Distributed Systems

In this chapter the DM pipeline is introduced and put into context of its application and of existing works. For this, first, the testing procedure of integrated systems is introduced, as this is a common scenario in which the proposed DM pipeline is used.

Integration Testing Cycle: The work flow during the development and testing phase of a large-scale distributed system is shown in Figure 3.1. There, the system is repeatedly tested to identify errors and potential for optimization, improved based on the gained knowledge and either released or passed to the next test cycle.
Such verification is usually performed on components, subsystems and the integrated system. In this thesis, the focus is on the integrated system which is formalized as a DMS.
A cost-efficient way to allow for large-scale verification of such integrated systems is to perform data-driven testing. For this, multiple test cases are performed on the integrated DMS that simulate the usage of the system under realistic user and environmental conditions. During those executions, trace data $\mathbf{K_b}$ is recorded and analyzed to verify correct system functioning.
This is challenging due to the following reasons. (1) The demand for high quality products leads to an increase of functionality implemented in such systems. Complexity of system interactions, as well as variability of system compositions grows. Those factors lead to more variants of the system behavior that need to be observed and verified during data-driven testing to allow for its verification. (2) This results in huge amounts of data with hidden and complex dependencies that need to be analyzed. (3) At the same time, the pressure of competitors requires to keep the time to market short, which is often achieved by keeping release cycles brief. That is, the product is tested extensively and iteratively within short periods of time. For example, within a period of a week, test

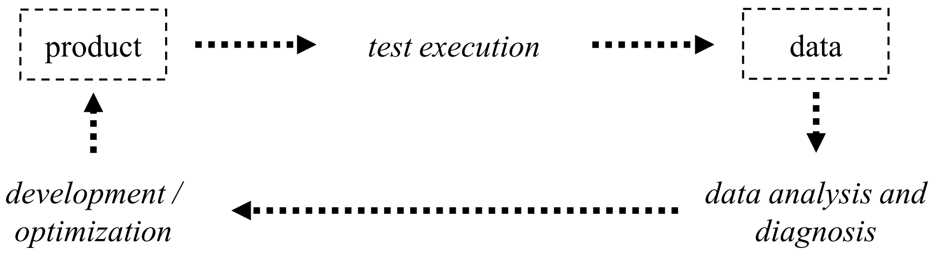


Figure 3.1: This image shows the product development cycle. In particular this includes recording and processing of data which is the focus of this work.

data is recorded by the test team and subsequently analyzed for faults and optimizations. Those are forwarded to the development team, which finds and corrects the errors or implements optimizations. This yields the next iteration of the DMS development cycle, which starts anew by recording data from the optimized system. With each iteration, components, subsystems, functional dependencies or interactions of the system might have changed, which makes data from previous iterations deprecated. Thus, next to the requirements of systematically analyzing complex and large amounts of data, efficiency of testing methods on fleets of objects is crucial.

Automated Diagnosis: To allow for this, (semi-) automated approaches are applied to handle errors by supporting experts in two main ways. First, this is through direct fault diagnosis, which includes finding, predicting, understanding and explaining errors. Second, this is through the improvement of the expert's understanding of the system, e.g. by extracting relevant information or by using appropriate visualizations.
Existing approaches are not ideally suited for this tasks, as will be discussed in the next section, which is why a novel DM pipeline is introduced here. This pipeline is intended to successively reduce complexity within the trace by finding correlating TVs and functional procedures. Those are modeled and used to infer specifications and dominant variants of behavior.

Chapter Outline: The state of the art of data-driven testing is discussed in the first part of this chapter. Foremost, this includes the field of failure diagnosis and existing methods for Specification Mining. Based on drawbacks of those approaches, the demand for a large-scale DM pipeline for DMSs is discussed.
Above that, the proposed approach operates within a data-driven testing work flow. Thus, to put this work into context, in the second part of this chapter, a systematic diagnosis work flow of modern large-scale data is presented. This flow provides an overview of procedures for diagnosis, including techniques for fault localization, RCA, anomaly detection and Specification Mining.
In the third part of this chapter, the proposed DM pipeline is introduced and discussed. Approaches that are required at each stage of this pipeline are presented, discussed and evaluated in successive chapters. Further, to validate the applicability and consistency of this method, in Chapter 9 a case-study is presented that is performed on a DMS of the automotive industry.

3.1 State of the Art

In this Section, the proposed Data Mining pipeline is put in context of existing research in the respective fields.

Overview: Fault Diagnosis in software of distributed systems has been an active field of research for many years. Approaches for diagnosis, of fields that are relevant to this work, can be mainly categorized into two types. First, those are methods to find and diagnose errors. Second, those are techniques that extract nominal behavior from data or that model it. This is either done to generate specifications that are relevant in model checking or for extraction of dominant behavior to improve understanding of

systems and data. The former includes, foremost, the fields of fault localization, root cause identification and anomaly detection. The latter includes automated failure/error diagnosis techniques for visualizing or modeling behavior and techniques for Specification Mining.
The proposed DM pipeline is closely linked to the former, as its basic procedure partly uses a well-established RCA procedure. However, unlike those approaches, the proposed method is used to systematically extract specifications and aims for semi-automated Specification Mining and extraction of dominant behavior from traces of a DMS. Thus, it can be equally categorized as a Specification Mining approach.
Consequently, in the following, first, a categorization and comparison of the proposed DM pipeline to approaches in diagnosis is performed and second, a comparison to other Specification Mining approaches is given.
Notably, this chapter focuses on the overall pipeline and literature related to it only. However, in further chapters, individual stages of this approach are discussed. There, related work on these stages is given in the respective chapters.

3.1.1 Modeling Behavior for Diagnosis

Definition: The proposed approach aims to both extract specifications from observed traces and to increase system understanding of complex DMSs by breaking down complexity and modeling observed behavior of subsystems. In this work, the latter is referred to as the extraction of dominant behavior, which is used to support diagnosis in three ways. First, by inspecting dominant variants of behavior that occurred before an error, potential causes may be identified. Second, by representing dominant variants of functional procedures that led to a specific target state (e.g. the shutdown of a car) system understanding is improved, e.g. the gained knowledge could be used to semi-automatically construct specifications. Third, by representing dominant variants of functional procedures across a trace, data understanding is improved, e.g. with there identifying procedures that took place at specific time spans within a trace.

This extraction of functional procedures in terms of both dimensions, procedures and dominant behavioral variants has been less researched in the past, while it has a high potential to significantly support an expert during data-driven diagnosis.
The most comparable approaches for this were presented in the field of automated failure diagnosis. The main aim of those techniques is to localize the most likely source of an error by performing RCAs [14]. Thus, in this section, the proposed method is put into context of those works on automated failure diagnosis and RCA. Those fields target to automatically identify root causes of failures, based on observed symptoms. Similar to the proposed approach, there, models are used to represent dominant and intended behavior for diagnosis to improve system understanding of the expert and to support him or her during diagnosis.
According to [14], such techniques can be categorized as rule-based, model-based, statistics based, Machine Learning-based, count-and-threshold based and visualization-based. The proposed DM pipeline can be used for extraction of such dominant behavior by aggregating functional procedures in models. Thus, it is categorized both as model-based and Machine Learning-based. According to the categorization that is used here,

the proposed approach is categorized as model-based, which is why here the focus is on literature on those types of approaches. Further, for the sake of completeness a short definition of the categories is given.

1. **Rule-based:** Rule-based techniques use rules to identify errors in the system. For instance, an expert might specify a directive that hints to the error, e.g. "if the car is faster than 10 km/h, while the engine is off". Limitations of such approaches include that unknown problems can not be detected and manual specification generation becomes in-feasible (as will be discussed further below) [14].
2. **Count-and-threshold:** Such techniques allow to discriminate between transient and intermittent faults. The further, refers to an internal system error that can be traced back to a system component, while the latter refers to an external error that cannot be traced back to that [14].
3. **Statistical:** Such techniques aggregate data of the system empirically. By using statistical tools such as correlations and histograms, experts can deduce potential problems and causes of those. In a broader sense, the TSCBN model that is introduced in Chapter 7 also models behavior of the data in terms of distributions. However, this is performed within a defined procedural model, which is why it was decided to categorize the proposed approach as model-based [14]. Here, statistics refer to empirical indicators that aggregate the system states. This is limiting as a more fine grained inspection in terms of procedures is not possible [14].
4. **Machine Learning:** The proposed DM pipeline is partly Machine Learning based. However, within this categorization in the context of diagnosis, this mainly refers to clustering approaches for anomaly detection or to identify odd behavior. The proposed approach, however, intends to derive behavior from a learned model, similar to the approaches categorized as model-based [14].
5. **Visualization techniques:** Visualization is a classical tool for the inspection of a system. In the field of DMSs often time-series analyses are used for inspection and to improve system and data understanding. Again with growing complexity this might become intractable. Other visualizations may visualize paths within a model. However, in contrast to the proposed approach, no automated extraction of dominant behavior is possible [14].
6. **Model-based:** In this field, models are used to represent either the nominal behavior of a system or a model of the observed behavior, which is similar to the approach that is proposed. Models might be of various types, such as physical, regression or graph-theoretic models. As part of the proposed DM pipeline in this approach a graph-theoretic model (more specifically a PGM) is learned from observed behavior.

Relevant model-based approaches in diagnosis
The proposed DM pipeline is composed of multiple steps, that ultimately yield a model that can be used for inspection of dominant behavior and for the extraction of specifications. Thus, it uses a similar analysis procedure, consisting of modeling and inference

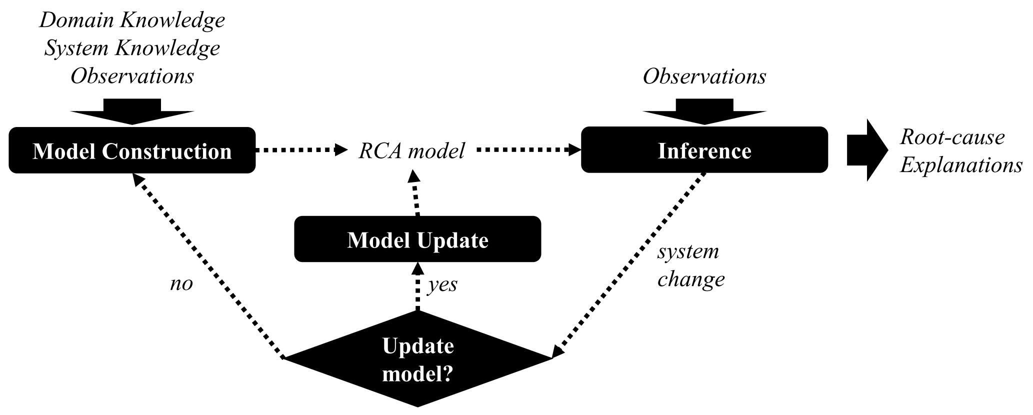


Figure 3.2: The procedure of RCA as it was presented in [2] is shown here.

steps, as it is common in RCA. However, in our approach we are focusing on DMSs and MSSs, which requires to develop additional steps, as well as new modeling and inference methods. In the following these three building blocks namely the analysis procedure, the models and the inference approaches are discussed in the context of this work.

Analysis Procedure: The basic procedure for model-based RCA is shown in Figure 3.2. There, a model of the problem is constructed using expert, system and observational input, which is then used to make inference (also called abduction) either on a new observation or on the modeled data. This yields an aggregated view on the data, which ultimately allows to find the root cause and an explanation of the analyzed problem. Such problems might be hardware or software based and are identified by analyses of the source-code or of data [2]. This RCA procedure may be surrounded by further preprocessing steps, such as the approaches of [19, 20, 21]. There, static or dynamic instrumentation is used to first, automatically capture identical procedures or profiles that occurred before an error and second, apply a RCA approach on those.
Depending on the target of diagnosis, different relations may be modeled. This includes, models that capture relations between nodes or components in terms of their topology. Also, this might be a set of symptoms, i.e. events, that indicate the error. The proposed approach has a target similar to the latter, as procedures of events are modeled, e.g. characteristic events before some target state (such as a failure). In that case, events within the model in the pipeline might be seen as symptoms. Multiple models might be used for that purpose, where each of those approaches is defined by a learning procedure. On the learned model, inference algorithms are used to deduce dominant or rare behavior [22].
The proposed approach consists of similar steps, i.e. a preprocessing, modeling and an inference step that identifies dominant behavior. However, in contrast to existing approaches it provides an end-to-end solution that is able to deal with peculiarities of DMSs and MSSs. In particular a specialized preprocessing approach is proposed, that allows to handle raw and large communication traces. Moreover, this method allows to break down complexity by identifying clusters in time and dimension, as well as by

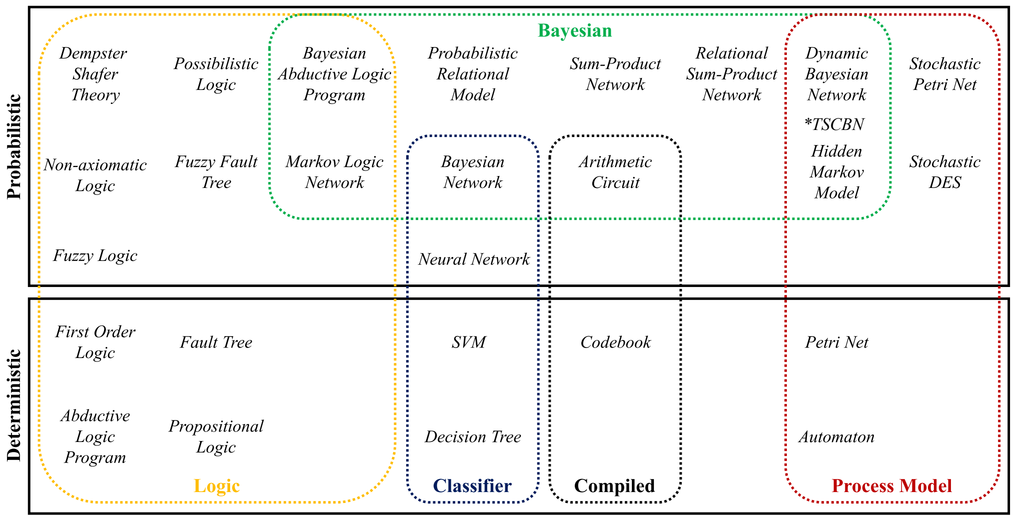


Figure 3.3: An overview of models used in previous works is shown as it was defined in [2]. In this work the TSCBN model is used (marked as *).

transforming traces to MSSs. With this, temporal behavior of subsystems of a DMS is inspected using a novel model and inference approach that is optimized for MSSs.

Analysis Models: In the RCA setting, multiple models are used for modeling and inference, none of which are used in the proposed pipeline, but rather, here, the TSCBN model is used that is specifically designed for MSSs.
According to [2] analysis models are either deterministic or probabilistic. The former does not capture uncertainty in the observed events or the inferences that are contained in the model, while the latter does capture this uncertainty [2]. An overview of such models is shown in Figure 3.3. The focus is on models that diagnose situations in which time of observation is relevant, i.e. where a sequence of events is considered, rather than static symptoms that indicate an error.
Multiple models were proposed for the representation of such dynamical behavior, the deterministic side of which includes, most importantly, process models, such as Petri Nets, or automata, such as Finite State Machines (FSM) [9]. This type of models captures each individual event as a node where succession of events is indicated by edges. This allows to capture many branches of behavior, but at the same time makes those models susceptible to quickly grow in size when learned from noisy data making it hard for the expert to interpret the model. Modern Process Mining approaches [23] and tools [24] allow to reduce this noise, e.g. by excluding rare events and performing diagnosis interactively. However, due to an exploding size still such models are particularly badly suited to be used for capturing behavior in large traces, as it is required in the given context of DMS traces.
Further models include classifiers such as Neural Networks or Support Vector Machines. This type of models is probabilistic in theory, while it is deterministic in the sense of reliability where exact inputs are mapped to exact outputs. Such models can classify a sequence in order to decide if it is erroneous or not, as well as parts within a sequence

to identify erroneous sequence elements. A problem with such models is its black box nature, which does not allow for direct expert input (except for hyper parameter tuning) or interpretability of internal procedures. Other classifiers such as Decision Trees allow to do such classification while being interpretable. However, for complex and noisy input data, this leads again to growth and highly complex structures making it less suitable in the given setting.
Models that are relevant in the context of this work are probabilistic white-box models which includes variants of dynamic Bayesian Networks, models relying on Markov Processes and models designed for special tasks. Those models are well-suited to be used for mining from complex traces, as those allow for compact representations, while capturing the uncertainty of observed behavior. That is why this type of models is used in the proposed DM pipeline. As the probabilistic modeling of MSSs of large-scale distributed systems is one step of the proposed DM pipeline, the related works section of Chapter 7 gives a detailed overview and comparison of those models.

Inference/Abduction Approaches: Based on models that were learned from recorded data, inference is performed to obtain the fault that generated the symptoms that indicate an error. In RCA, this is done in many ways [14], e.g. by inferring how problems propagate through the system [25, 26, 27, 28], by finding the error source or by representing how successes propagate through the system [29, 30] to then compare failure cases against success cases. Also, this is achieved through systems that capture expected behavior and flag a problem whenever this behavior is violated. In contrast to this, in the proposed DM pipeline the aim is different. The pipeline models the process of MSSs that occurred within certain functional procedures using TSCBNs and infers its most likely behaviors from it as dominant system states. Technically this is similar to modeling propagation through the system and analyzing symptoms.
Above that, techniques for inference vary with the type of model that is used for training. In the proposed DM pipeline, Bayesian Networks (BNs) are used for this purpose, which is why this is mainly discussed here. For BNs, techniques are marginals that are used to find the state with highest likelihood that led to an error [31, 32], which is solved using approaches such as Junction Trees [33], Node Elimination [34], Adaptive Inference [35], Loopy Belief Propagation [36] or Markov Chain Monte Carlo approaches [32]. Further, this includes the Most Probable Explanation (MPE) method that returns the most likely constellation that was present in the error case using MPE bucket tree [37] or SLS [38]. For the case that a subset of values of RVs is given, the Maximum A Posteriori (MAP) estimate is found using approaches such as stochastic sampling [39].
In the proposed pipeline, the modeling and inference part uses a similar approach. There, a representation of a set of MSSs is learned as a variant of a BN and then, Gibbs Sampling is used to determine the MPE. In that sense, this part of the proposed DM pipeline differs mostly in the type of model that is used, which in contrast to existing approaches captures a multidimensional snapshot of the temporal functional procedure in state and time. This includes representation of symptoms to infer dominant states in order to reveal constellations that are characterizing an error.
In other types of models, inference is done by using symptom vectors [40] and similarity [41] to identify problems. Unlike our approach, this excludes temporal dependencies making it less precise.

Further methods are designed to extract most likely behavior or error locations in specific models and thus, its usability in the current setting is limited by the complexity and model capacity of the model. In Petri Nets the most likely sequence are extracted, e.g. using the Viterbi puzzle [42]. In Decision Trees, heuristic explanation methods [44] or simplification methods are used, such as looking for locations where the most faults occurred [45]. In Fault Trees approaches include tree search [46], while in Markov Logic Networks those are Logic abduction [47, 48] or exploitation of marginals [49, 50]. Such methods, again do not allow to infer behavior in terms of temporal dependencies between dimensions, making those less precise when applied in DMSs that are represented as MSSs. In the case of classification, a learned classifier might be used to infer if a test sample is erroneous or not [43]. Such approaches, unlike the proposed approach, however, require known error cases for training.

Automated diagnosis approaches
As shown in Figure 3.2, RCA involves similar steps as the proposed pipeline, which is modeling, and inference based on observations. However, those form only a subset of the proposed approach, as the proposed method additionally includes systematic preprocessing and identification of functional procedures, as well as a focus on inference of specifications and dominant behavior. That is why, next, this work is put into context of related automated mining procedures that are used in fault diagnosis. Categorization of those approaches is not clearly possible as those consist of multiple steps. Nevertheless, here a grouping based on the type of the main step in the respective pipelines is given.

Signature-based: This groups' methods require known errors which are provided or identified in a previous step. Those are the following. The approach in [51] collects traces with static instrumentation and uses classifiers, which are trained on a set of known problems, to determine the error type. As this type is known it corresponds to the root cause. In [54], system logs are inspected by monitoring and collecting traces of correct executions, which are, then, used to generate a FSM that generalizes the collected traces and perform failure analysis on it. A method that requires to first identify common behavior is described in [55], where signatures of system behaviors are extracted and clustered based on a purity score. With this, error locations are identified and Tree Augmented BNs are used to determine metrics that correlate with anomalous periods.

Similarity-based: Approaches of this type group similar sequences given a similarity metric. In [19] anomalies are automatically detected by identifying identical procedures of similar activities for fail-stop and non-fail-stop cases. This is done using a dynamical instrumentation approach. These activities are compared using distance metrics to identify segments that are substantially different from the others followed by a ranking. Those segments are then inspected for their root cause by comparing segment elements. The work of [20] similarly collects run time profiles, but uses static instrumentation. Those profiles are clustered together to be presented to an expert for inspection of the root cause. Pinpoint [52] records traces from applications and determines if an error occurred using either known faults or by using probabilistic context-free grammars if faults are unknown. Then, clustering and decision trees are used to identify erroneous behavior. In particular by inspecting the decision tree, root causes are found.

Outlier Detection: Another type of automated approaches are those that find outliers for the identification of errors. Magpie [21] collects traces of events which it clusters together, identifies elements that are far from a cluster as outliers and models the event sequences in a probabilistic manner. Event transitions of low probability are then, considered potential root causes. The method of [53] aims to identify node-level anomalies in large-scale systems. For this it groups nodes and represents each of those as features, which are then used to identify outliers. Those outliers are inspected by an expert to identify faulty nodes. To automatically infer root causes in messages, the approach of [15] applies a hybrid log analysis approach to identify and cluster similar parts of a trace and to perform outlier detection using message flow graphs. Additionally, a model checking approach is integrated in this flow that allows to identify known errors in traces.

Model-inference-based: Approaches that are similar to the proposed approach learn models which are used for inference. The approach in [56] infers behavioral models from observed legal executions and compares failing executions with the inferred models to automatically identify the likely anomalous events that caused observed failures. In [57] event correlation mining is presented. There, logs are analyzed by automatically preprocessing events, mining events, extracting rules between events and extracting a graph (called Event Correlation Graph), which are used for successive error inspection tasks.

Probing approaches: Another way to perform diagnosis is to probe or simulate a system behavior to learn valid behavior. The method of [43] uses defined test cases to produce a trace under intended scenarios. Those are clustered according to good and bad performance. Classification approaches are used to identify rules that indicate causes with this.

3.1.2 Extraction of Specifications

3.1.2.1 Specification Mining

Definition: Verification of program execution is done by defining specifications, which are used to to find specific errors. Such specifications are formalized in multiple ways including rules, state machines, automata or specification languages such as LTL or Computational Tree Logic (CTL). However, a major drawback of this is the manual formalization of such specifications. Therefore, in 2002 the field of Specification Mining arose. This field deals with the automated learning of specifications using Machine Learning, which was first introduced by Ammons et al. [58].
By leveraging automatically learned specifications, unknown errors are found and verification on fleets of complex systems, that contain similar functional procedures, is performed. Those characteristics make such approaches especially useful for diagnosis of integrated DMSs during their testing phase, as high complexity hinders manual definition of specifications, unknown and known bugs are hard to find manually and fleets need to be tested efficiently, e.g. by checking learned specifications weekly.
Current algorithms for inference of specifications are categorized as static [59, 60, 61, 62, 63] or dynamic [58, 64, 65, 66, 67, 68, 69, 70, 71, 72]. As the proposed DM pipeline

is a dynamic mining approach, the focus of the following overview is on the latter type of mining.

Static mining: Those approaches infer specifications from program code. As the program's ground truth is known such methods are very accurate. However, with increasing program complexity, those algorithms do not scale, due to rising numbers of program branches [73].

Dynamic mining: Such miners extract specifications from simulation or execution traces, such as the ones extracted from the DMS. This is especially required if the program code is not available or system complexity is too large. In a DMS, especially the latter is the case, as multiple interacting components, many TVs and many functional procedures are present.
Dynamic approaches have the advantage that coarseness of recording can be configured to fit required needs and run time information (e.g. user inputs or run time data types, are inherently available). Also, in contrast to static mining, dynamic methods do not need to consider redundant or impossible implementation paths and can mine specifications from traces independent of the implementation's programming language.
However, the quality of mined specifications depends on the quality of the input trace set and completeness of specifications depends on the fraction of observed program paths in a trace. In the testing phase of complex integrated DMSs this is less of a problem, as various defined test cases are performed on the respective system to maximize the fraction of observed paths.
Current dynamic miners extract specifications as finite state automata [58, 64, 65, 66], invariants [67], temporal logic formulas [68, 69], timed regular expressions [70] or further temporal properties [71, 72].
Dynamic miners can be mainly categorized into two types, which are model-based and non-model-based approaches.

Model-based: Ammons et al. [58] first dealt with the inference of specifications from program execution traces. In their work they mine probabilistic finite automata (PFSA) representing temporal as well as data dependencies under the assumption that the system under analysis is mostly correct. This approach was extended by Lo and Khoo [65] by filtering out erroneous traces and clustering related traces. Per cluster a Finite State Automaton (FSA) is inferred. These FSAs are then combined to a larger FSA that embraces all cluster automata. FSAs that satisfy binary temporal properties of three different types are found by the approach in [64], which improves precision through refinement and coarsening. Such approaches produce automata, that are hard to interpret and whose complexity increases with growing functionality. This, in contrast to the presented method, aggravates expert input.

Non-model-based: Yang et al. [71, 72] mine two-event temporal patterns from execution traces. Their tool, Perracotta, infers instances of eight common two-event patterns, e.g. the alternating pattern (xy)*. These properties can then be chained together to form larger rules. However, Perracotta misses out some rules and thus delivers only partially complete specifications [65]. Also, a single violation of a pattern prevents it from being

mined, thus, requiring Perracotta to have perfect traces. Javert [66] also uses chaining rules to construct more complex specifications from simpler patterns. Perracotta is extended in [72] to find temporal properties for the analysis of digital hardware designs. Their work mines four binary patterns and uses inference rules to produce more complex properties. These approaches differ from the presented miner in this work, such that the proposed approach mines temporal rules of arbitrary length and does not rely on chaining to construct more complex specifications from simpler ones.
Response patterns between sequences of events are found using sequential pattern mining in [69]. In [67] Daikon is presented, which mines invariants of values of program variables, e.g. X_0 denotes that the variable X is always positive. Lemieux et al. [68] introduced Texada, which finds instances of user-provided LTL property templates of arbitrary complexity. However, this only allows to find predefined patterns and misses out properties of any other structure.
Lastly, in the related field of temporal assertion mining approaches were proposed for verification of hardware designs [74, 75, 76]. Such approaches often exploit system design knowledge and tend to produce hard-to-read properties, as opposed to the more structured representation of the proposed approach with TVs as dimensions and its values.

3.2 Research Demand

Based on the given state of the art, this work has two main aims, which cannot be sufficiently handled for large-scale distributed systems with existing methods. Those are the identification of functional procedures, as well as the extraction of dominant behavior and specifications from traces of large-scale distributed systems in a (semi-) automated manner.

Identifying dominant behavior: As described in Section 3.1.1 frameworks for anomaly detection and fault diagnosis group similar activities to successively use RCA approaches including modeling and inference to perform diagnosis. This approach is similar to the DM pipeline that is proposed in this work. Further, the pipeline performs offline analyses on a large trace that consists of multiple functional procedures that occur within it. Unlike existing approaches, here, one main part of this work deals with the scalable identification and processing of functional procedures, while existing approaches assume those to be given. This is challenging due to high dimensionality and length of traces.
In addition to that, other approaches aim to analyze symptoms to analyze errors, while this work aims to identify dominant functional behavior for improved system understanding at a large-scale. This is important, as in modern working environments experts might change frequently and hidden correlations make it impossible to manually identify dimensions and functional procedures that are part of a function and thus, of interest to the expert. Thus, a semi-automated way (as the one introduced here) of providing insights into large multidimensional system data is of high importance and only less considered in the context of diagnosis so far.
Moreover, large-scale distributed systems produce heterogeneous data on a system state level, while existing approaches mostly assume homogeneous ones. Thus, to include all types of data during the analysis the proposed DM pipeline performs a defined set of

data preprocessing steps to unify data formats automatically.
Another difference to those approaches is, that TSCBNs are used to capture system behavior. This model is especially well suited to represent functional procedures in terms of MSSs. It inhibits the procedural structure and thus, provides a more expressive representation, which allows to provides more accurate snapshots of dominant behavior in the data yielding more precise specifications.
Also, as stated above MPE is used for inference, which is similar to existing approaches. But in contrast to those approaches, rather than a snapshot of symptoms, multiple snapshots in time and state are extracted from functional procedures in the data.
Lastly, existing frameworks do not consider raw traces in large-scale distributed systems and do not have to ensure scalability. The proposed DM pipeline includes this aspect and provides an effective way of preprocessing to reduce the data at an early stage. This reduction comes at the cost of losing event information, which however, is tackled by providing a model that can handle the resulting MSS format. This format still contains enough information to extract meaningful knowledge from it.

Specification Mining: In terms of Specification Mining existing approaches do not address the following aspects.
First, those do not assume multiple functions, but rather assume data to be given from a defined function.
Next, those are unable to find properties of arbitrary length, e.g. by relying on techniques such as chaining of events. In contrast to that, the given approach constructs a TSCBN per functional procedure, which is flexible in length as it solely depends on the length of the given functional procedure.
Due to the lack of functional separation, especially in large-scale distributed systems existing approaches also tend to exhibit higher false positive rates as random cooccurrings of different functions might be misinterpreted as part of the same functional procedure. This effect is reduced here by identifying relevant functional procedures first.
In addition to that many miners assume perfect traces that are used to deduce nominal behavior. This is especially difficult to obtain in the scenario considered here, where the testing stage of a system is considered which may contain a high number of faulty components. This makes them inapplicable for Specification Mining from imperfect traces that contain multiple functions.
This is amplified in large-scale distributed systems, as heterogeneous traces of massive size are produced. This is solved by including appropriate preprocessing of data in terms of cleaning, homogenization and transformation.
Above that, in the considered DMS MSSs are modeled as TSCBN, which captures dimensional dependencies and correlation between TVs in a procedural and compact manner. This allows to reduce complexity, such that expert input can be included. Further, with this specifications are found as a multidimensional snapshot of behavior (e.g. by using MAP on TSCBNs), rather than a linear chain of events. With this, e.g. whole MSSs could be verified in terms of likelihood rather than as series of events.

Proposed Solution: To address this research demand, a systematic DM pipeline is developed, that allows for dynamic mining of specifications and identification of dominant behavior in large-scale distributed systems. In particular, the following challenges of existing approaches are addressed with this.

- **Multi functionality:** Existing approaches are extracting correlations within a trace, based on all events that those contain. This is sufficient if the trace consists of a single functional procedure. However, for the case of DMSs multiple functional procedures might exist. Those need to be identified and separated both in dimension and time and are not known a priori. To solve this, the first stages of the pipeline deal with a dimensional and procedural segmentation and clustering that identifies functional procedures within traces.
- **End-to-end processing:** Instead of starting from a trace of one function, the designed pipeline is an end-to-end solution, that automates the process of Specification Mining from the recorded raw trace to a set of resulting specifications of multiple functional procedures. In particular, this includes the steps of data reduction, interpretation and unification which are important to handle heterogeneous and large traces.
- **Semi-automated mining:** An optimal approach would be able to learn specifications fully automatically. However, due to hidden correlations and a high amount of meta knowledge that is hard to represent, this is currently not possible. Meta knowledge includes e.g. steady states that influence a procedure, while those are only rarely observed together with it. Also, this includes naming of entities or the distinguishing between data points in a trace that relate to behavior rather than to other aspects such as communication headers. To handle this missing information, in this work a semi-automated process is proposed for this. There, expert input is included both during grouping of TVs and functional procedures, as well as during structure discovery of TSCBNs.
- **Include structural information:** Further, as opposed to other works information about the temporal structure is included by modeling data as TSCBNs. Thus, instead of deducing specifications from correlations between events, dedicated multidimensional functional procedures of MSSs are identified as a basis for this. This allows, to both include expert knowledge and to exploit the inherent structure of a DMS (i.e. state vector $\mathbf{X}$). For example in the case of concurrent events the order of events might alternate in the trace, making it hard to asses its correct ordering on an event basis, while modeling the structure allows to handle this.
- **Imperfect traces:** Further, the proposed approach assumes systems that are potentially in its development phase, i.e. extracted traces are imperfect, which means that noise and erroneous behavior might be present. For example, permanent errors might be in the trace, which make it hard to simply extract specifications in terms of its most frequent behavior. This is handled at multiple stages in the DM pipeline. First, it is assumed that multiple instances of DMSs are considered of which a subset contains the correct behavior. Second, this approach applies preprocessing on the trace, such that a clean, homogeneous and structured representation is found. Third, the proposed TSCBN model models causality of MSSs only between events of TVs, which do correlate. Fourth, by filtering for relevant

TVs and correlating functional procedures the pipeline allows to reduce the number of possible false positive associations between unrelated TVs and its state changes.

- **Arbitrary length:** The proposed approach does allow for traces of arbitrary length as functional procedures can be of arbitrary length. Thus, by representing such procedures as TSCBNs resulting specification are restricted by the model's length only. As the length of functional procedures only depends on the length of the segment set that is to be modeled and those segments might be of arbitrary length, the proposed DM pipeline is able to mine specifications of arbitrary length.

- **Computational Complexity:** Further, large data sets are assumed here, which are recorded from multiple instances of a DMS. Scalable approaches are required and complexity needs to be broken down successively, which is solved here by targeting the approach towards defined functional procedures instead of mining specifications from the whole trace.

- **Concurrency:** DMSs are concurrent in nature, which requires to extract specifications from multiple possibly interacting functional procedures. This is harder than processing sequentially executed data. TSCBNs are used for this purpose, as those allow to capture dimensional and procedural information in state and time. This allows for an expressive representation of functional procedures, which in turn results in expressive specifications.

3.3 System Work Flow

In this section a modern diagnosis work flow that is commonly used during the testing phase of a large-scale distributed system is described. The proposed DM pipeline needs to be designed such that first, it is applicable within this type of work flow and second, challenges of those diagnosis frameworks are taken into account. That is, why in this section those systems are introduced and discussed.

In order to perform diagnosis at a large-scale, automated methods are required to process vast amounts of data, that are produced by many instances of systems. This needs to be done within short time spans and with high test coverage. Therefore, in modern diagnosis Big Data frameworks are used for this purpose. Such frameworks consist of multiple components and are implemented on multiple machines with large memory and high processing power. Figure 3.4 shows possible components that are used within such systems [3]. There, incoming data is extracted, preprocessed, stored, analyzed and knowledge inferred, which is then, visualized and provided to an analyst or a relevant business unit.
Automated flows of diagnosis are implemented on such infrastructures. A modern flow, that is commonly used in data-driven testing of modern systems is proposed here and discussed in the following. Its basic building blocks are shown in Figure 3.5.

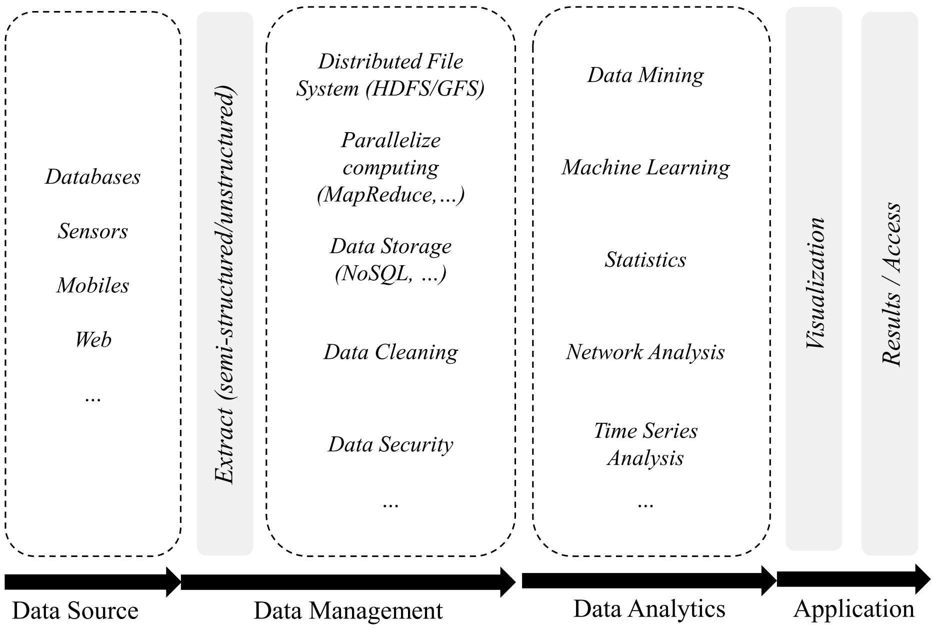


Figure 3.4: An example of a big data system with its components as it was presented in [3].

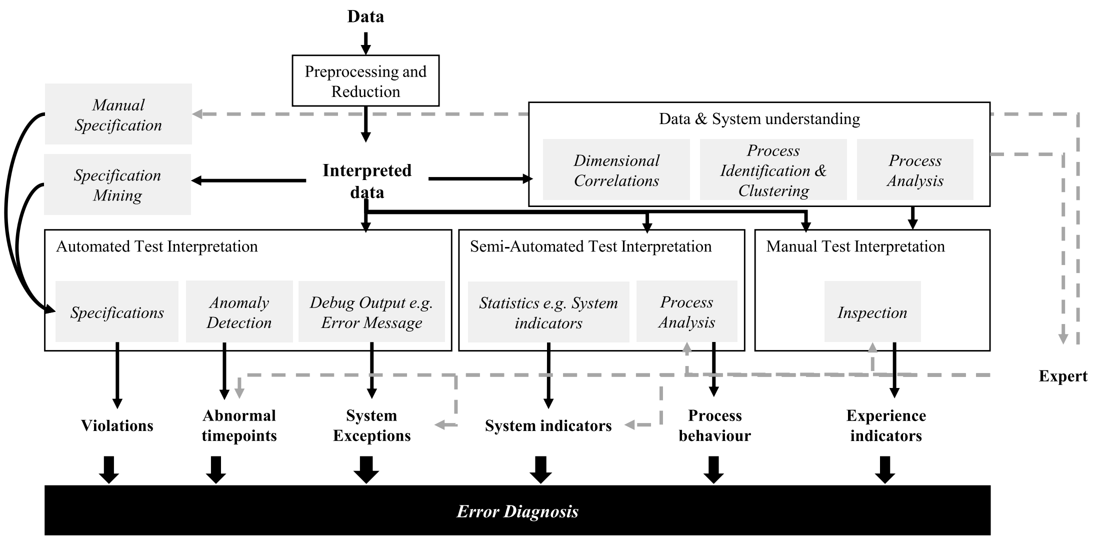


Figure 3.5: A modern system work flow for diagnosis is shown that includes multiple sub procedures.

3.3.1 Targets of Big Data Frameworks

Typically diagnosis frameworks include systematic fault detection, fault localization, root cause identification, as well as explanation of known and unknown errors. Solution of those tasks necessitates the following requirements.

- **Reduction, Redundancy and Missing Structure:** Recorded data is often provided in an encoded manner in order to reduce storage costs, e.g. as a sequence of bytes. Such data needs to be interpreted and reduced towards aspects that are relevant for diagnosis of the application under test. This becomes especially complex with growing data size, which requires parallel interpretation and reduction approaches. Further, the interpreted trace data might contain redundancies, which need to be removed and is in general composed of TVs with heterogeneous data types, which need to be structured to provide a uniform format.

- **Data Understanding:** Data is recorded in an unstructured manner, while structure in terms of functional procedures (see Chapter 2) is inherently present in the data. This can be used to reduce the search space in which an expert needs to look for errors. For the diagnosis of such functional procedures only a subset of data in both dimension and time is relevant for analysis. Thus, to increase understanding in such data, inspected data is structured and compacted. This is achieved by identifying functional procedures with its corresponding TVs.

- **System Understanding:** In today's verification ultimately, correctness of verification depends on the experience of the experts, who need to be familiar with valid variants of functional executions. This gets increasingly complex as subsystems are evolving, growing in complexity and are designed by defined domains only. Especially, when added to an integrated system, hidden interactions and unexpected paths of executions might occur. This requires a high amount of experience. Thus, to allow experts to inspect and learn about behavioral variations from patterns, automatic approaches to represent complex executions of relevant sub-functions are used. For example, this might be white-box models that aggregate the data.

- **Systematic Analyses:** Data is ingested frequently and often becomes deprecated once an iteration of optimization is finished. Therefore, analyses need to be performable weekly in a standardized manner, such that no redundant processing on the trace is run. This is solved by application of automated DM pipelines for systematic diagnoses. For example, for the application of error localization, diagnoses could be classifiers that classify error types in the data or unsupervised approaches that identify anomalous samples. Also, this might be the automated learning of specifications which might consequent be applied on unknown traces which allows to systematically test data from fleets.
  In addition to that, redundant processing of data needs to be reduced in Big Data. Thus, steps within the framework need to be modular. That is, those steps need to provide results that can be used for multiple consequent approaches and that allow for expert input at any stage of the process.

- **Computational Tractability:** The processing of data needs to be computationally tractable. That is, scalability needs to be ensured, which includes reduction of memory consumption in storage and during computation, as well as efficient and parallel implementations.

3.3.2 Systems Work Flow

The proposed modern automated diagnosis work flow of Figure 3.5 is defined as follows.

Preprocessing: First, in a systematic preprocessing step, recorded traces $\mathbf{K}_b$ are interpreted and reduced, e.g. by removing redundancies.

Data and System Understanding: Recorded data is used to support the expert in better understanding the data and the system. This is achieved in multiple ways, such as through identification of correlating dimensions, through identification of groups of procedures or through aggregation of procedures in appropriate models.
To do so, similar functional procedures [19] are inspected at this stage. This is done with anomaly detection methods [19] to identify odd data points (e.g. by highlighting those in the trace), RCA techniques to identify correlations (e.g. by highlighting probable paths in a model) or process mining approaches such as replay [23] (e.g. by replaying data on a process model [23]).

Systematic Analyses: Various approaches are applied on the reduced data for the purpose of diagnosis. Those are used to perform localization, explanation or detection of unknown and known errors and are categorized as automated, semi-automated or manual in the following.

- **Automated methods:** First, this includes systematic checking of specifications on all incoming recorded traces, which allows to locate known errors. This is especially useful in large-scale distributed systems, as it allows to identify many known errors in an efficient manner. Classically those specifications are designed manually by experts. However, with growing complexity of systems this becomes intractable, which is why automated Specification Mining approaches are used to learn specifications from data or program code.
  Second, exceptions are printed to the debug output and thus, forms a part of the recorded traces. Those entries contain the error information and thus, identify known errors in the trace. This allows to directly deduce the cause of the error by interpreting the type of exception that was recorded.
  Third, this includes localization of unknown errors. For this, anomaly detection methods are used which are able to extract candidate errors. Experts or ranking approaches then, filter and classify those according to severance or add them to the collection of known errors.

- **Semi-automated methods:** Such approaches require expert inputs during its execution.
  First, expert input can be included by extraction of system indicators that might point towards spots of misbehavior in the system, e.g. the time it takes for a car to shut down. When, this value is identified abnormal by an expert an error is found. Further, this value might already yield the error type and an experienced

expert might be able to directly explain the error from this.
Second, the system might be broken down into relevant functional procedures of the system. By analyzing those, e.g. using inference algorithms or methods from Process Mining (PM), bottlenecks or rare behavior is identified. This again requires active investigation of the resulting representation by the expert in order to make conclusions about both error location, type and explanation.

- **Manual methods:** Those methods work on the raw information contained in the data and rely solely on expert knowledge. That is, an experienced expert is familiar with relevant indicators, expected system executions and potential hot spots that might contain an error and its cause. Here, exploratory data analysis tools are used for diagnosis. Those might be tools for visualization of time-series or tools to investigate descriptive statistics of the data.

As discussed in Section 3.1, many exiting approaches target individual aspects within this work flow. In this thesis the focus is on two particular paths of those. First, this is the automated extraction of specifications and second, this is the increase of system understanding by identifying functional procedures and extracting dominant behavior from those. Existing methods do not allow to sufficiently solve this task in the context of large-scale distributed systems as was pointed out in Section 3.1. That is why, in the next section a novel semi-automated DM pipeline is presented for this.

3.4 Data Mining Pipeline

In this section an overview of the proposed DM pipeline is given.
First, the definitions of Chapter 2 are revised and put in the context of the analysis intent of the DM pipeline. After this, an overview of the proposed approach is given, before an overview of its individual stages is given in successive sections.

3.4.1 Assumptions

In Chapter 2 the concept of functions and functional procedures was introduced, which represents the overall behavior of a DMS. All functions F_k input and modify the current state $\mathbf{X}^i$. In case that the function is correct any input constellation $\mathbf{X}^i$ yields the expected output constellation. In systems of low complexity, correct states of functional procedures f_{ki} of function F_k are exactly known by the experts and thus, are used to manually design specifications. In a DMS this would require to represent all possible input output constellations of this function for any $\mathbf{X}^i$. This is complex, as the dimensionality of $\mathbf{X}$ is often high with multiple states per dimension, e.g. 1000 dimensions with 3 states already lead to 3^{1000} possible input combinations for F_k that need to be modeled. Therefore, it is essential to reduce complexity by performing diagnosis not on the total state $\mathbf{X}^i$, but rather per functional procedure and the subset of dimensions of $\mathbf{X}^i$ that are relevant to it. This reduction is performed with the following two assumption.

- **Decoupling:** The consequent state of any subsystem $\mathbf{R}_x \in \mathbf{R}_{sub}$, depends on all functions F_k that operate on $\mathbf{R}_x$. Thus, functional procedures of the corresponding functions that operate on common subsystems after integration are coupled, while in reality such functions are often developed and tested by separate experts. In a correctly functioning system coupling of functions should not influence each other and thus, correct functional procedures result. Therefore, it is assume that when the functional procedure works mostly correct, all functions F_k can be independently considered and bad behavior considered noise. Consequently, testing of behavior of individual functions is done separately.

- **Vertical Reduction:** Further, for certain functions F_k only a subset of actions $\mathbf{Q}_{sub}$ and environmental influences $\mathbf{W}_{sub}$ are of relevance. In the correct case similar $\mathbf{Q}_{sub}$ and $\mathbf{W}_{sub}$ yield similar functional procedures. With this, complexity is further broken down. Moreover, it is assume that there is a low number of functional procedures f_{ki} that is active within any short time spans.

It is assumed that in the correct case all functional procedures result in similar behavior in state and time when similar conditions are given. Further, in the error case variants of that functional procedure might be possible. To capture the dominant correct and the inferior erroneous behavior, it is proposed to assume each functional procedure f_{ki} to be a distribution in state, time and procedure.
Each distribution of a functional procedure f_{ki} can be interpreted as a data generator, that produces instances of a certain functional procedure depending on the currently prevalent conditions (i.e. system states $\mathbf{X}$). Also, each instance of a functional procedure f_{ki} is a sample from the corresponding data generator.
Resulting from this, any observed trace can be seen as a result of sampling from a collection of functions and from the corresponding functional procedure distributions, resulting in functional procedures that are active at overlapping time spans.

System correctness: With those assumptions a DMS $\mathbf{R}$ is correct if all observed instances of functional procedures f_{ki} of all functions F_k are correct.

Goals: As a result the two main tasks of a verification pipeline are to (1) identify samples of functional procedures and to (2) learn its distributions (e.g. using a TSCBN). Based on those distributions, it is expected that more likely behavior is observed more often and the maximum a posteriori estimate of the according procedural distribution is likely to correspond to nominal behavior.

3.4.2 Proposed Diagnosis Pipeline

An overview of the proposed DM pipeline for semi-automated Specification Mining and deduction of dominant behavior is given. The main idea is to successively break down complexity to identify functional procedures which are, then, analyzed to extract specifications and dominant behavior. As illustrated in Figure 3.6 this can be achieved with the following steps.

1. Preprocess traces to yield a reduced, interpreted and uniform MES. Traces of multiple system executions are used as an input.

2. Identify correlating TVs of functional procedures.

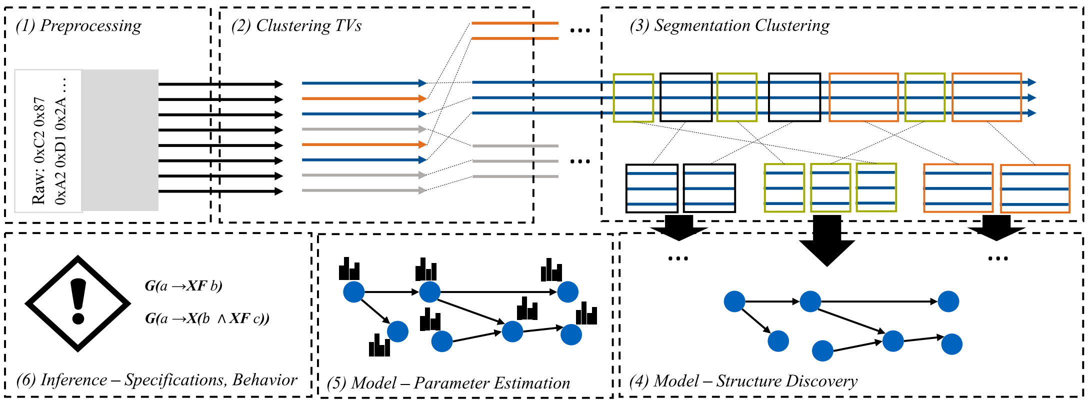

(1) Preprocessing
Raw: 0xC2 0x87 0xA2 0xD1 0x2A …
(2) Clustering TVs
(3) Segmentation Clustering
(4) Model – Structure Discovery
(5) Model – Parameter Estimation
(6) Inference – Specifications, Behavior
G(a →XF b)
G(a →X(b ∧XF c))

Figure 3.6: An overview of the proposed pipeline is shown. This approach consists of six main steps. These are preprocessing (1), clustering of TVs (2), segmentation and clustering (3), learning of model structure (4) and its parameters (5) followed by inference of specifications and dominant behavior (6).

3. Identify functional procedures and its correlating instances in time.
4. Learn the structure of each functional procedure from its observed instances.
5. Learn the corresponding distribution of each functional procedure.
6. Deduce specifications and dominant behavior per functional procedure by extracting behavior of high likelihood from its distribution.

In the remainder of this chapter these stages are introduced. To allow for a consistent application of the pipeline, methods at individual stages were extended and compared to baseline approaches. This is also discussed here.

3.4.3 Input data

The input data of the framework is a raw trace $\mathbf{K}_b$ that is transformed to a MES with the properties specified in Chapter 2. Further, depending on the task, data of the according format might be ingested at any of the first three steps. Among others possible inputs are the following:

- **Erroneous traces:** If the dominant behavior is error free such traces are modeled as functional procedures under noise.
- **Error-free traces:** Those are modeled directly as functional procedures.
- **Known target segments:** To identify dominant behavior that occurred directly before a failure or before a target state change, segments before each failure (or target state) instance are modeled as functional procedures.
- **Segments of similar execution:** To increase understanding of variants of procedures or to identify misbehavior in such variants, multiple executions of a certain functional procedure are analyzed.

Note that each output is systematically extracted from a fleet of objects and directly used as input for the proposed DM pipeline.

3.4.4 Preprocessing

If required, the raw input trace $\mathbf{K}_b$ is converted to its MES format $\mathbf{K}_s$ and reshaped to form a MSS $\mathbf{K}_n$, that is used as an input for the consequent processing steps. As described in Chapter 2 this trace might be unstructured, contain redundancy and have heterogeneous data types.
Thus, at this stage good data quality needs to be established in terms of data completeness (e.g. missing entries), accuracy (e.g. include vs. exclude noise) and consistency (e.g. illogical entries). Notably, in diagnosis the degree of quality required, depends on the task to be performed. For exact tasks such as fault detection, missing entries might be errors and noise needs to be included. However, for tasks, such as Specification Mining, such noise needs to be filtered out and a more consistent representation is desirable.
Here, data is transformed and cleaned, the trace format is unified and redundancy is removed. In general $\mathbf{K}_b$ might be of any format that encodes MES.

Processing: The input to this step is a raw trace $\mathbf{K}_b$, which is preprocessed to yield an interpreted MES $\mathbf{K}_s$ and a reduced MSS $\mathbf{K}_n$.

$$\mathbf{K}_n, \mathbf{K}_s = prep(\mathbf{K}_b) \tag{3.1}$$

Extension and Evaluation: An important group of traces $\mathbf{K}_b$ are encoded messages, e.g. as found in network traces of communicating distributed systems. Such traces are of large-scale and therefore, require an efficient approach for preprocessing. In [4] an interpretation and reduction strategy was introduced, which is revised in Chapter 4. Evaluation of this stage was performed on a set of in-vehicle network traces.

3.4.5 Clustering Temporal Variables

Traces in DMSs are high dimensional, while vertical correlations between TVs exist, which is used to identify the vertical dimension of functional procedures. In general these groupings are not known in advance. Also, it is intractable to identify these manually as dimensionality may be large, the integrated system is developed by multiple domains and a high degree of experience is required.
Also, functions F_k and its functional procedures f_{ki} operate on subsets of TVs of the trace ($\mathbf{K}_n$, $\mathbf{K}_s$) only. During diagnosis domain experts need to focus on groups of TVs that are relevant to them only, e.g. for the analysis of the active cruise control of a car TVs such as the state of the wiper are irrelevant.
As part of this pipeline, unsupervised Machine Learning algorithms in combination with expert input are used to identify correlating TVs that are part of common functions F_k. This requires expert input, as clustering granularity (i.e. hyper parameters of the clustering approaches) is dependent on the target of clustering.
Thus, this stage reduces complexity by adding structure to the data and by revealing vertical correlations in it.

Processing: The input to this step is an interpreted trace $\mathbf{K}_s$, a clustering approach C, and hyper parameters of the clustering approach P_C. The output is an assignment $a(S_i)$ of TVs S_i to K defined clusters $a(S_i) \in \{1, ..., K\}$. The operation performed is

$$a = sigcl(\mathbf{K}_s, C, P_C) \tag{3.2}$$

Each cluster $k \in \{1, ..., K\}$ together with the expert selection allows to reduce the trace $\mathbf{K}_s$ in dimension, resulting in a trace $\mathbf{K}_s^k$ that has only TVs $\hat{S}$ that were chosen by the expert, potentially as part of a certain cluster assignment $\hat{a}$.

$$\mathbf{K}_s^k = sel(\mathbf{K}_s, \hat{S}, \hat{a}), \text{ where all } S_i \in \hat{S} \text{ and } a(S_i) = \hat{a} \tag{3.3}$$

$$\mathbf{K}_n^k = sel(\mathbf{K}_n, \hat{S}, \hat{a}), \text{ where all } S_i \in \hat{S} \text{ and } a(S_i) = \hat{a} \tag{3.4}$$

Notably K might be 1, i.e. one cluster with all signals could be chosen. Moreover, the MSS $\mathbf{K}_n^k$ is further processed, as it provides a reduced and more meaningful representation of the functional procedures (e.g. no redundancies).

Extension and Evaluation: Clustering is used at this stage. As there is no clear definition for the correlation of TVs, similarity metrics, that are used to identify groups of TVs, need to be designed towards the target of grouping.
For this, in Chapter 5 a feature-based clustering approach is described that was first introduced in [5]. There, occurrences in common time spans are used as a metric for correlation of TVs.
This approach is evaluated on multiple traces $\mathbf{K}_s$ that were recorded from cars of a big OEM.

3.4.6 Segmentation Clustering

Multiple instances of functional procedures f_{ki} occur in each trace $\mathbf{K}_n^k$, which need to be found at this stage. At the same time, segments in the trace that correspond to similar functional procedures need to be identified. For this, an automated segmentation and clustering approach is required.
The granularity of functional procedures depends on the target of diagnosis. Here, for the target of Specification Mining, segments of high similarity are preferable, as the aim is to identify dominant correct functional procedures.
With this stage data is horizontally structured and correlations in terms of similar functional procedures are revealed.

Processing: The input to this step is a trace $\mathbf{K}_n^k$ with a reduced set of TVs, a segmentation approach D and hyper parameters of the segmentation approach P_D. The output are multiple sets of MSSs $\hat{M}$, each representing a set of MSSs M_i of a functional procedure. The operation performed is

$$\hat{M} = segcl(\mathbf{K}_n^k, D, P_D) \tag{3.5}$$

Notably D might include a clustering approach that allows for expert input.

Extension and Evaluation: To allow for discovery of MSSs, existing segmentation approaches are extended in Chapter 6. Further, alternative segmentations are discussed. Also, to allow for expert input in terms of granularity, at this stage clustering is used to further subgroup segments.
These approaches are compared on a synthetic data set.

3.4.7 Model - Structure Discovery and Parameter Estimation

At this point $\hat{M}$ contains segment sets M_i of instances of defined functional procedures. Those segments might be of high dimensionality and complexity, which is why here functional procedures are aggregated into appropriate models. In general those models depend on the focus of diagnosis and e.g., might be Process Models, Probabilistic Graphical Models (PGMs) or supervised classification models (e.g. Decision Trees, Deep Neural Networks (DNN)). With this step an aggregated representation of the data is extracted.

Processing: The input to this step is a set of MSSs $M_i \in \hat{M}$, a model type T, a learning approach L and the hyper parameters for learning P_L. The output is a learned model Q. The operation that is performed is

$$Q = learn(M_i, T, L, P_L) \tag{3.6}$$

Extension and Evaluation: For the task of Specification Mining the model Q needs to allow to represent the nominal behavior of a functional procedure. In particular this includes capturing the temporal structure of MSSs under uncertainty in state, time and procedure. In [8] a model was presented for this, which is described in Chapter 7.
The model, its structure discovery and parameter estimation approaches are evaluated on a synthetic data set.

3.4.8 Inference of Specification and Behavior

Based on the learned models, inference is performed to get insights for diagnosis. For the given goal of specification extraction, this includes the identification of likely states of the learned distributions of functional procedure that are represented in TSCBNs.

Processing: The input to this step is a trained model Q, an inference approach I and the hyper parameters for inference P_I. The output is the gained knowledge W, which depends on the inference performed, e.g. this could be a list of specifications or most dominant states. The operation performed is

$$W = inf(Q, I, P_I) \tag{3.7}$$

Extensions and Evaluation: TSCBNs are used to extract LTL specifications that resemble potential specifications. An approach to this was first presented in [10] and is revised in Chapter 8.
This approach is evaluated both on a synthetic data set and in a real world example.

Above that, the MPE is used to identify dominant behavior which is also discussed in this chapter.

3.4.9 Evaluation of Framework

The evaluation of approaches at each individual stage is presented at the respective chapter. Next to this, the overall pipeline is evaluated on five real world application scenarios from the automotive industry.
This is done to evaluate the applicability of this approach in real-world scenarios, of hyper parameters per stage, of consistency of the pipeline and of the quality of its outcomes. The results of this are presented in Chapter 9.

3.5 Summary and Conclusion

In this chapter a DM pipeline for large-scale distributed systems is introduced. This approach intends to systematically extract specifications and dominant behavior from traces recorded from integrated systems. Existing approaches are not end-to-end, are not capable of mining heterogeneous traces, do not handle multi-functionality, do exclude structural information of the distributed system, do not include expert input, do not sufficiently handle imperfect traces or cannot produce arbitrary length specifications. The proposed approach tackles those challenges by introducing a five step procedure, which systematically homogenizes and reduces the data, identifies functional procedures, models those and performs inference on those. At the same time expert input is included at multiple steps along the pipeline. For this, existing approaches are extended or compared at each stage, in order to allow for a consistent flow through this pipeline. These approaches are presented in the following chapters. Additionally, the evaluation of the framework is presented in a case study in Chapter 9.

4 Automated Interpretation and Reduction of Traces at a Large Scale

In this Chapter the focus is on the first step of the proposed DM pipeline, which is the automated preprocessing of trace data $\mathbf{K}_b$ recorded from executions of large-scale distributed systems. The input to this step is a raw trace $\mathbf{K}_b$, which is preprocessed to yield an interpreted trace $\mathbf{K}_s$ and a reduced trace MSSs $\mathbf{K}_n$, by performing

$$\mathbf{K}_n, \mathbf{K}_s = prep(\mathbf{K}_b) \tag{4.1}$$

This preprocessing step depends on the input data format of such traces which could have manifold shapes. This includes directly recorded system states $\mathbf{X}$ such as recordings from simulations where all states are known at all times. Also, this might be the debug log that is output on individual computational devices. Further, those are encoded state sequences, which is a common type of data that is obtained from the system at a low cost. This type of data is recorded in internal networks of automotive or IoT by recording the internal messages that are transmitted for communication. Especially in the case of automotive traces those messages transmit all relevant system state information as well as sensory input that senses both interaction with the system and environmental states.
This forms an important group of traces, which is why here the focus is on this type of raw traces. Those traces are represented as an event sequence of encoded system states and are used in systematic analysis work flows as the one presented in Chapter 3. This is due to the fact that this representation allows to reduce the storage costs of the recorded trace. However, as states are encoded the data is also hard to access for analysis and thus, requires dedicated preprocessing approaches before being applicable for further processing.
In particular, recorded raw trace data has several characteristics that need to be addressed during preprocessing. Those are the following.

Data Characteristics: The main characteristics of the input data include the following. Traces $\mathbf{K}_b$ are in a raw data format that is redundant, encoded, of massive size (i.e. $> 10\ 000\ 000$ samples), of high dimension (i.e. > 3000 TVs of $\mathbf{R}$, $\mathbf{Q}$ and $\mathbf{W}$), heterogeneous, unstructured, noisy and contains multiple functions.

Requirements: The task of Specification Mining requires the traces to only contain the essence of what is relevant to represent the nominal behavior of a system. Further, to be able to include expert input, the mining procedure is run per domain of an expert. That is, only functions that are relevant to a domain need to be considered. To allow for those two tasks, first, a reduced representation needs to be found that is targeted towards the analyzing domain by extracting relevant TVs. Second, redundancies need

to be removed and a homogeneous representation of the data needs to be found.
In addition to that, as part of the automated DM pipeline presented in Chapter 3, this preprocessing needs to be deterministic and automated towards systematic extraction of expert relevant output. Moreover, as data is of massive size and data is processed as batch, a scalable reduction and interpretation approach needs to be integrated.
Thus, in terms of Specification Mining good data quality is achieved by homogenization of TVs, reduction of noise per TV, reduction of trace size and by extraction of domain specific aspects of the data that allow to be ingested in further mining steps. Thus, both an accurate and consistent data set needs to be found at this stage.
Existing approaches do not allow for automated interpretation and reduction of encoded large traces, which is why a framework for this is presented. This framework was first introduced in [4] in the context of automotive and is used in this thesis as a preparation step within the automated Specification Mining pipeline.

Chapter Outline: First, in Section 4.1 related works are presented and in Section 4.2 the framework of [4] is detailed. Next, in Section 4.3 benefits of this procedure in the context of Specification Mining are discussed. Lastly, in Section 4.4 existing evaluation results from [4] are presented together with additional experiments.

4.1 Related Works

As little comparable frameworks exist, related works include those on preprocessing of similar types of data, which is in-vehicle networks, and those on user-aided trace analysis frameworks.
A work similar to the framework of this chapter is introduced in [77], where vehicle data is collected from test vehicles at a large scale and analyzed by experts for faults. However, they focus on rule-based and visual error-diagnosis from data subsets, while the focus here is on preprocessing of task-specific data from full-system traces.
Using in-vehicle network traces for Data Mining was part of several prior works. CAN signals are used for predictive maintenance by mapping signal groups and finding relevant signals using wrapper methods [78], by extracting signal characteristics at various times, using histograms [79] or for fault detection, using condition indicators [80]. CAN signals are used for road type classification [81] and driver workload monitoring [82]. In [83] detection of driver distraction from signal features is presented. Detecting faults with CAN signals was proposed early in [84] and in [85], where signals are partitioned into segments and features extracted. In terms of preprocessing the authors of [86] introduced a clustering approach for vehicular-sensor data to optimize it for Data Mining, which unlike the preprocessing here, groups data for reduction. Similarly to the introduced work, in [87], SAX [88] is used to symbolize numeric signal values for motifs relevant for diagnosis. In [89], the authors aggregated vehicle signal data, such as average speed, to predict compressor faults in trucks and in [90] to model the remaining useful life time in trucks. In contrast to the presented approach, signal preprocessing in those works focuses on aggregated data or small subsets of numeric signals, which does not raise the problem of interpreting massive traces. Others extract features to either minimize computational cost for on-board applicability or to optimize it towards a specific Data Mining task, resulting in loss of generality for other Data Mining tasks.

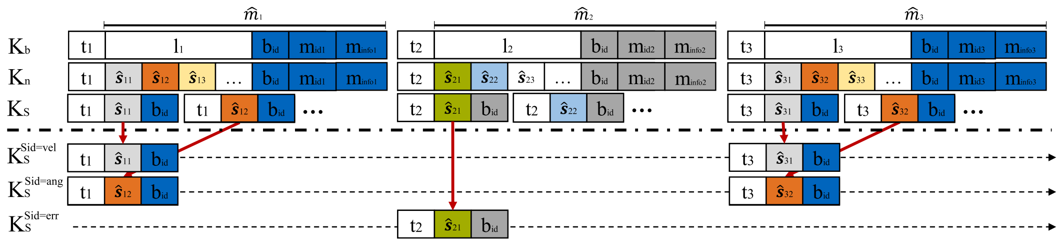


Figure 4.1: $\mathbf{K}_\mathrm{b}$ is the recorded raw trace. Its payload l_i contains certain TV types that are defined by $m_{\mathrm{id}i}$ and $b_{\mathrm{id}i}$. TV instances with same $m_{\mathrm{id}i}$, $b_{\mathrm{id}i}$ and position l_i form one source of information, such as $\mathbf{K}_s^{s_\mathrm{id}=vel}$, which is a discrete time-series describing the vehicle speed or $\mathbf{K}_\mathrm{s}^{s_\mathrm{id}=err}$ which is a temporal sequence marking time-instances were a certain error was sent. In this example the velocity TV instance sequence $\mathbf{K}_\mathrm{s}^{s_\mathrm{id}=vel}$ is extracted by taking the messages sent at t_1 and t_3 and interpreting their first value to get e.g.$\hat{s}_{11} = (10, vel)$ and $\hat{s}_{31} = (12, vel)$ for speed 10 at time t_1 and speed 12 at time t_2. E.g. in a BMW Series 7 2 million messages are generated for $\mathbf{K}_\mathrm{b}$ per minute.

However, the presented framework extracts a task-specific and general representation of data without feature extraction, while determining features could be a subsequent part to the presented process.
In the context of distributed systems, in [91], an approach to remove irrelevant events from event traces in Web application logs based on constraints is presented and in [92], user-based data simplification in a medical context was introduced. In [93], anomaly and in [94], intrusion detection in large scale network traces is presented. In the software context, users are involved for processing event traces with Data Mining [51] and in optimizing software design [66]. To detect faults based on event traces, several user-involved analysis frameworks are proposed. They include approaches for application failures [19], for problem detection in internet services [52] and to examine execution reports for error detection in clusters [20].
Those works focus on analyses aimed towards certain Data Mining tasks and with or without user interactions for simplification. By contrast the framework used here is an automated approach.

4.2 Automated Interpretation and Reduction Pipeline

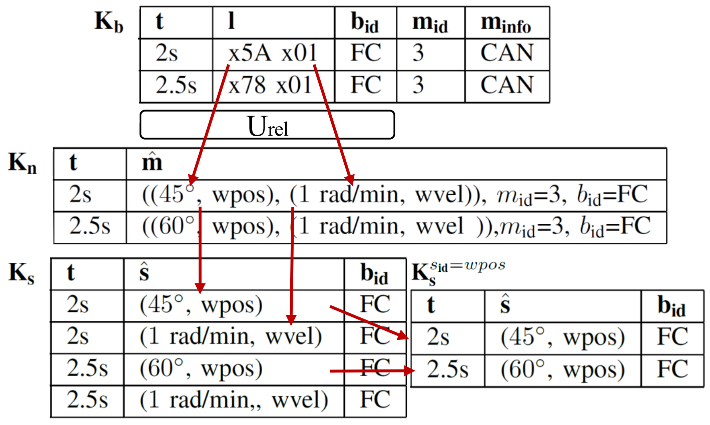
4.2.1 Overall Processing Pipeline

The framework [4] aims to preprocess data that is initially encoded in messages of format $\mathbf{K}_\mathrm{b}$. Those are transformed to $\mathbf{K}_\mathrm{n}$ and $\mathbf{K}_\mathrm{s}$, which can then be used to consider the extracted data per TV with name S_{id}. These data types are formally introduced according to 2 in the following and the concept of how those types correlate are shown in Figure 4.1. Oftentimes devices communicate with each other, in order to transmit state information of multiple TVs. With this functionality between those is established and provided, e.g. in in-vehicle networks this might be messages exchanged for the wiper function. The transmitted information is encoded in messages, which have dedicated state information sent in dedicated message types. Those messages might be sent on different communication channels and thus, be of different protocols. Also, the transmitted state information includes both sensory and actuator data of the distributed system. Such traces are recorded directly from the communication channel, successively analyzed offline and used for various diagnosis tasks, e.g. in this case for Specification Mining. This raw trace forms $\mathbf{K}_\text{b}$ as presented in Chapter 2 in this scenario. As this data is recorded from communication channels, additional formalizations to the ones given in Chapter 2 are required to understand the overall proposed pipeline. This formalization is introduced here.

Formal Definitions: The communication between devices is performed via messages of type m, where each message has a unique identifier m_id. Any message with m_id transmits the exact same type of information, i.e. the states of a defined set of TVs. Further, a message instance $\hat{m}$ is referred to as an occurrence of a message type m in the trace. In the same way it is referred to the type of a TV S as s which is identified by an identifier s_id, which could be the unique name of a TV. The occurrence of this TV in a message is declared as $\hat{s}$. The set of all types of TVs that may exist in a message are $\Sigma = \{s_1, s_2, ..., s_z\}$. Each instance $\hat{m}$ of a message type m contains an exactly defined set of TV types $\bar{S} \subseteq \Sigma$ and the channel on which it occurs b_id. Thus, the message type can be written as

$$m = (\bar{S}, m_\text{id}, b_\text{id}) \tag{4.2}$$

, where the number of state types transmitted $|\bar{S}|$ can vary per message. Further the TV instance consists of an identifier s_id that defines the TV type and a value v it has



t	l	b_id	m_id	m_info
2s	x5A x01	FC	3	CAN
2.5s	x78 x01	FC	3	CAN

t	$\hat{m}$
2s	((45°, wpos), (1 rad/min, wvel)), m_id=3, b_id=FC
2.5s	((60°, wpos), (1 rad/min, wvel)),m_id=3, b_id=FC

t	$\hat{s}$	b_id
2s	(45°, wpos)	FC
2s	(1 rad/min, wvel)	FC
2.5s	(60°, wpos)	FC
2.5s	(1 rad/min,, wvel)	FC

t	$\hat{s}$	b_id
2s	(45°, wpos)	FC
2.5s	(60°, wpos)	FC

Figure 4.2: Wiper function: Example of the formalization used. Assuming l' to be the first two bytes (=wpos) per message and l'' the last two bytes (=wvel) the rules for mapping between $\mathbf{K}_\text{b}$ onto $\mathbf{K}_\text{n}$ are $v = 0.5 \cdot l'$ and $v = l''$ [4].

at that time, which corresponds to the state of the TV at that time. In the case of general communication systems TVs might be concerning either a function (e.g. steering angle), a control unit (e.g. reset) or the network (e.g. frame qualifier). For the case of Specification Mining the focus is on the functional TVs here.

Example: A good example that visualizes this concept is given in Figure 4.2. There instances of all messages of type m' identified by $m_{\text{id}} = 3$ contain all TV types $\bar{S}' \subseteq \Sigma$ related to the wiper function, which is sent on a communication channel called FC, which has an identifier $b_{\text{id}} = \text{FC}$. The set of contained TVs is $\bar{S}' = (s_{\text{wpos}}, s_{\text{wvel}})$ has s_{wpos} which include the TVs wiper position and s_{wvel} wiper velocity. Further, an particular instance $\hat{m}'$ of the message type m' has the instances $\hat{m}' = ((\hat{s}_{\text{wpos}}, \hat{s}_{\text{wvel}}), 3, \text{FC})$ with $\hat{s}_{\text{wpos}} = (45°, wpos), \hat{s}_{\text{wvel}} = (1\ \text{rad/min}, wvel)$.
Further for this scenario the data format is declared as

$$\mathbf{K}_{\text{b}} =< k_{\text{b}1}, k_{\text{b}2}, ..., k_{\text{b}w} > \tag{4.3}$$

with $|\mathbf{K}_{\text{b}}| = w$, byte tuple $k_{\text{b}j} = (t_j, l_j, b_{\text{id}j}, m_{\text{id}j}, m_{\text{info}j})$ with l as message payload in byte format, b_{id} as channel identifier, m_{id} as message identifier and m_{info} as protocol specific message fields used for protocol specific translation. For instance in a CAN bus m_{id} is the CAN identifier.
Further the payload l can be translated to its TV instances it contains using its unique identifier m_{id}. Thus, after interpretation $\mathbf{K}_{\text{b}}$ can be written as:

$$\mathbf{K}_{\text{n}} =< (t_1, \hat{m}_1), (t_2, \hat{m}_2), ...(t_w, \hat{m}_w) >=$$
$$< (t_1, (\hat{S}_1, \hat{m}_{\text{id}1})), (t_2, (\hat{S}_2, \hat{m}_{\text{id}2}), ...(t_w, (\hat{S}_w, \hat{m}_{\text{id}w}) >$$

By extracting individual instances $\hat{s}$ from the set of TV instances, per time of occurrence, this can be written as:

$$\mathbf{K}_{\text{s}} =< (t_1, \hat{s}_{11}, b_{\text{id}1}), (t_1, \hat{s}_{12}, b_{\text{id}1}), ...(t_w, \hat{s}_{w1}, b_{\text{id}w}), ... >$$

Per TV, the identifier s_{id} can be used to find all $\hat{s}_{ij}$ instances of the same TV from its messages, which allows those to be grouped. That is each TV represents one state information of the system. For instance to obtain the velocity of the wiper as information it suffices to filter for the according identifier s_{id}, which yields the temporal state sequence of the according TV $\mathbf{K}_{\text{s}}^{s_{\text{id}}=wvel} = \sigma_{s_{\text{id}}=wvel}(\mathbf{K}_{\text{s}})$. With this formalization the preprocessing approach allows to extract an interpreted and domain-specific homogeneous representations from traces that are initially recorded from a large-scale distributed systems. This step enables automated Specification Mining to be performed based on this. The overall approach of [4] is shown in Algorithm 4 using relational algebra and in Figure 4.3 as an overview that includes the main steps of the approach.

4.2.2 Overview

The automated framework yields a reduced, interpreted and domain-specific representation of traces from large distributed systems. Its main steps include structuring, interpretation and reduction, as shown in Figure 4.2 and Algorithm 4.
For early reduction and minimal interpretation cost, in line 3 of Algorithm 4 task-dependent TV types are preselected from $\mathbf{K}_{\text{b}}$ based on domain knowledge. In particular

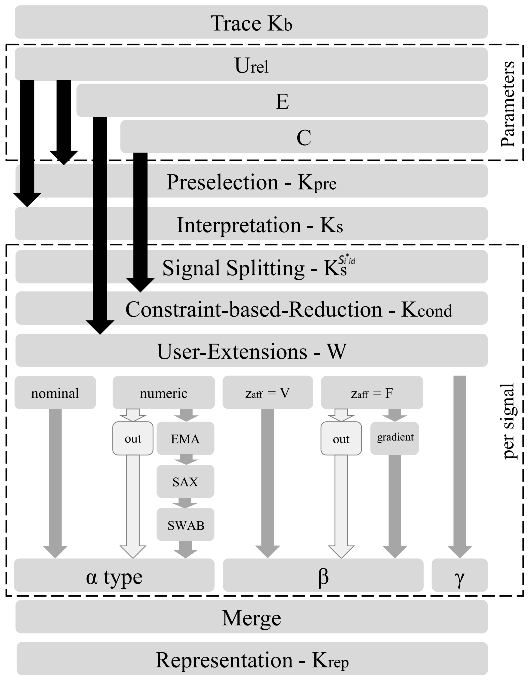


Figure 4.3: Overview of the flow of the preprocessing framework [4].

for the task of Specification Mining this is done by providing a list of TVs that might be related to the specification to mined. That is, detailed clustering and extraction of related TVs is done in the clustering step of the framework proposed in this thesis, while here a selection of the expert is included. That is, for better efficiency experts can exclude irrelevant TVs at this stage, which could have potentially yielded erroneous or false positive specifications. Further, this exclusion is imperative as computational complexity is reduced by this, which is important to handle massive trace sizes.
$\mathbf{K}_{\text{pre}}$ is interpreted by looking at payload bytes of $\mathbf{K}_{\text{pre}}$'s elements that contain TV types relevant for analysis (line 5) and map those bytes of payload l in $\mathbf{K}_{\text{b}}$ to a TV instance $\hat{s}$ (line 6) resulting in $\mathbf{K}_{\text{s}}$. Knowledge of relevant bytes per TV type and their positions are contained in the interpretation rules U_{rel}. Also, only relevant message instances $\hat{m}$ are inspected.
Next, as TV instances vary in type, each TV type s (e.g. with $s_{\text{id}} = vel$) in $\mathbf{K}_{\text{s}}$ is processed individually as a sequence $\mathbf{K}_{\text{s}}^{s_{\text{id}}}$. Per TV type, in lines 10 to 11, domain-specific reduction rules are used to reduce data to relevant data points. To extend each data point with meta-data or pre-calculated values in line 11 extension rules are defined. Also, varying types of TVs require prior classification of TV data types and type-dependent preprocessing (lines 13 to 28). They specified three classes (α, β, γ) of types. For each class they aim to obtain a symbolic representation of the data that reduces size while preserving outliers relevant for fault diagnoses. While this is useful in general diagnosis, for the case of Specification Mining outliers might be excluded at this step in order to reduce noise. For numeric values they determine a representation in terms of trend extracted using the SWAB approach presented in [95] and range found using SAX as

Algorithm 1 Algorithm taken from [4]

Reduction, Interpretation and Homogenization of raw Byte sequences
Input: *trace* $\mathbf{K}_b$*, preselected interpretation rules* U_{rel}*, reduction rules* C*, extension rules* E
Output: *homogeneous, reduced, interpreted Sequence* R_{out}

```
1: $R_{\text{out}} = \emptyset$                                                         ▷ Output
2: $U_{\text{comb}} \subseteq U_{\text{rel}}$                                             ▷ Preselection
3: $\mathbf{K}_{\text{pre}} = \sigma_{(m_{\text{id}}, b_{\text{id}}) \in U_{\text{comb}}}(\mathbf{K}_{\text{b}})$
4: $\mathbf{K}_{\text{join}} = \mathbf{K}_{\text{pre}} \bowtie_{\mathbf{K}_{\text{pre}}.b_{\text{id}} = U_{\text{rel}}.b_{\text{id}} \cap \mathbf{K}_{\text{pre}}.m_{\text{id}} = U_{\text{rel}}.m_{\text{id}}} U_{\text{rel}}$       ▷ Interpretation
5: $\mathbf{K}_{\text{join2}} = \mathcal{F}_{u_1}(\mathbf{K}_{\text{join}})$
6: $\mathbf{K}_{\text{s}} = \mathcal{F}_{u_2}(\mathbf{K}_{\text{join2}})$
7: for each $s_i^* \in \Sigma^*$ do                                                ▷ TV Splitting
8:     $\mathbf{K}_{\text{s}}^{s_i^*{}_{\text{id}}} = \sigma_{\mathbf{K}_{\text{s}}.s_{\text{id}} = s_i^*{}_{\text{id}}}(\mathbf{K}_{\text{s}})$
9:     $(\mathbf{K}_{\text{sep}_i}, \mathbf{K}_{s_{\text{cor}}}^{s_i^*{}_{\text{id}}}) = e(\mathbf{K}_{\text{s}}^{s_i^*{}_{\text{id}}})$
10:    $\mathbf{K}_{\text{cond}_i} = \mathbf{K}_{\text{sep}_i} \bowtie_{\mathbf{K}_{\text{sep}_i}.s_{\text{id}} = C.s_{\text{id}}} C$                 ▷ Reduction
11:    $\mathbf{K}_{\text{red}_i} = \sigma_{\mathbf{K}_{\text{cond}_i}.e = true}(\mathbf{K}_{\text{cond}_i})$
12:    $W_i = \mathcal{F}_E(\mathbf{K}_{\text{red}_i})$                                      ▷ Extension
13:    $\tau = type(\mathbf{K}_{\text{red}_i}, Z)$                          ▷ Type-dependent processing
14:    if $\tau$ is $\alpha$ then
15:        $(\mathbf{K}_{\text{num}_i}, \mathbf{K}_{\text{nom}_i}) = typeSplit(\mathbf{K}_{\text{red}_i})$
16:        $(\mathbf{K}_{\text{num}i_{\text{out}}}, \mathbf{K}_{\text{num}_{i\text{rep}}}) = outlier(\mathbf{K}_{\text{num}_i})$
17:        $\mathbf{K}_{\text{num}_{i\text{clean}}} = m(\mathbf{K}_{\text{num}_{i\text{rep}}})$
18:        $\mathbf{K}_{\text{res}_i} = \mathbf{K}_{\text{num}_{i\text{out}}} \cup \mathbf{K}_{\text{num}_{i\text{clean}}} \cup \mathbf{K}_{\text{nom}_i}$
19:    end if
20:    if $\tau$ is $\beta$ then
21:        $(\mathbf{K}_{\text{F}_i}, \mathbf{K}_{\text{V}_i}) = functionSplit(\mathbf{K}_{\text{red}_i})$
22:        $(\mathbf{K}_{\text{F}_{i\text{out}}}, \mathbf{K}_{\text{F}_{i\text{clean}}}) = outlier(\mathbf{K}_{\text{F}_i})$
23:        $\mathbf{K}_{\text{F}_{i\text{clean}}} = addGradient(\mathbf{K}_{\text{F}_{i\text{clean}}})$
24:        $\mathbf{K}_{\text{res}_i} = \mathbf{K}_{\text{F}_{i\text{out}}} \cup \mathbf{K}_{\text{F}_{i\text{clean}}} \cup \mathbf{K}_{\text{V}_i}$
25:    end if
26:    if $\tau$ is $\gamma$ then
27:        $\mathbf{K}_{\text{res}_i} = \mathbf{K}_{\text{red}_i}$
28:    end if
29:    $R_{\text{out}} = R_{\text{out}} \cup \mathbf{K}_{\text{res}_i} \cup W_i$                             ▷ Merge
30: end for
```

described in [88] (line 17). This is a trade-off between reduced data size, that can be handled during analyses, and loss of exact value information. However, by extracting outliers (line 16) prior to symbolization important information for fault analysis is preserved. Lastly, the processed and symbolized data is merged together in line 29, forming a state representation R_{out} that can be used for Specification Mining. These steps can be split in a interpretation and a reduction phase, which are detailed separately in the following sections.

4.2.3 Interpretation Phase

For efficient Specification Extraction functions need to be considered in a semi-supervised manner by domain-specific experts that perform the extraction. For this step multiple TVs are relevant, which need to be extracted. This is done with the presented approach as it allows for an automated and parameterizable way for each domain to extract representations with a defined reduced number of TVs. This TV extraction process is presented here. There, this is done by per-domain, extracting a reduced trace $\mathbf{K}_{\text{s}}$ that contains only a subset of specified TVs from the raw trace $\mathbf{K}_{\text{b}}$. In particular this is done on massive traces which requires processing on Big Data frameworks. However, such approaches require to operate in a distributed manner, which requires the extraction operations to be formulated in a tabular manner. TVs relevant for Specification Mining are extracted by specifying relevant TVs once and then, running the defined framework as batch for any incoming data set. Comparable tools [96] parse sequentially through all messages of a trace and lookup each TV per message, while the approach discussed here uses a distributed approach for this. This makes that approach fit well in the proposed Data Mining work flow of Chapter 3, as it is designed to enable automated domain-specific large-scale extraction of relevant aspects on large fleets of objects.
To achieve this first, relevant TVs to extract need to be specified as an input to that framework. With this the approach is able to obtain a extracted and interpreted version of relevant TVs, which is particularly efficient as interpretation cost is kept low by filtering relevant messages at an early stage, i.e. prior to interpretation.

Structuring and Preselection
Experts of a function are usually aware of all subsystems and components that are involved, which allows those to coarsely specify a subset of TVs that have to be included in the mining process, although detailed interactions and particular TVs might be not known. Extracting such hidden relations will be discussed in Chapter 5.
It is inefficient to translate all TV instances in all message instances. But, as this approach allows to provide a subset of TVs it is possible to only consider a relevant subset of bytes per message. This allows to do the following.

Structuring: The relevant TVs are specified as a set $U_{\text{comb}} \subseteq U_{\text{rel}}$ of translation tuples u_{rel}, where the set U_{rel} contains all tuples of possible TVs. This is used in the approach for efficient retrieval of TV values $\hat{s}$ from $\mathbf{K}_{\text{b}}$. For instance if the function to be mined is the Wiper, an expert might specify the position wpos and the velocity wvel of the wiper as relevant TVs in U_{comb}. Also, he might include action TVs such as pressing the handle bar for wiper activation or environmental TVs such as the rain intensity. For this to work a set of meta information needs to be passed as translation tuple, which is

$$u_{\text{rel}} = (s_{\text{id}}^{\text{rel}}, b_{\text{id}}, m_{\text{id}}, u_{\text{info}}) \tag{4.4}$$

, where u_{info} contains interpretation rules, that define how the encoded information is to be translated, $s_{\text{id}}^{\text{rel}}$ has the ids of TVs that are relevant to the mining process (e.g. wpos and wvel), while b_{id} and m_{id} contain the according channel and message ids on which the TV occurs.

Preselection: It is preferable to perform less operations directly on $\mathbf{K}_{\text{b}}$. To do this, in the first step the knowledge of relevant message ids, resulting from the expert specification of TVs, can be exploited to filter $\mathbf{K}_{\text{b}}$ for all m_{id}s and b_{id}s of TVs in U_{comb}. For instance one could select U_{comb} as the TVs wpos and wvel in Figure 4.2, which reduces the rows in $\mathbf{K}_{\text{b}}$ to only entries that have a $m_{\text{id}} \in (3)$ and $b_{\text{id}} \in (FC)$. WIth this, only a byte sequence of relevant message types $\mathbf{K}_{\text{pre}}$ remains. A simple example was presented in [4] for U_{rel} and is given in Table 4.1. Notably, the tuple of relevant entries U_{rel} has an identical schema as U_{comb}.

$\mathbf{s_{id}^{rel}}$	$\mathbf{b_{id}}$	$\mathbf{m_{id}}$	$\mathbf{u_{info}}$
wpos	FC	3	Int.rule: $v = 0.5 \cdot l$; rel.B = l' = (1,2)
wvel	FC	3	Int.rule: $v = l$; rel.B = l' = (3,4)
wtype	K-LIN	11	Int.rule: $v = l + 2$; rel.B = l' = (1)
wstat	SOME/IP	212	Int.rule: $v = l$; rel.B = l' = (10,22)

Table 4.1: Example for U_{rel} with relevant bytes to extract: Bytes 1 and 2 for wpos in messages with id 3, Bytes 3 and 4 for wvel. From SOME/IP the wiper status wstat and from K-Lin the wiper type wtype could be extracted from messages with respective ids 11 and 212, i.e. the presented approach allows to combine multiple protocols into this extraction [4].

Information Interpretation

Interpretation is an expensive operation and needs to be performed on relevant bytes only. Further, for scalability reasons it is required to state the problem in terms of scalable database operations, as processing frameworks for Big Data, such as Apache Hadoop [18] are optimized to work on those effectively. Moreover, the format should enable row-wise extraction of TV instances with interpretation rules u_{rel}. This is solved by doing the following.

Interpretation Rule: Mapping $\mathbf{K}_{\text{pre}}$ to $\mathbf{K}_{\text{s}}$ can be performed by applying a mapping u on each element k_{b} of $\mathbf{K}_{\text{pre}}$ for each u_{rel} that is associated to any k_{b}. Extraction information u_{info} needs to contain the byte positions at which the TV value v can be found in the payload l of k_{b} (see Figure 4.1) and how it can be evaluated to the TV value v. This includes the definition of condition based positions, i.e. rules where values of preceding bytes define the presence of a TV type in succeeding bytes. Furthermore, evaluation information need to be given, such as data types, coding, protocol based fields and translation rules (e.g. intercept to add or mapping of a Hex to a categorical value). After having specified relevant TVs in U_{comb} the uninterpreted version of the trace $\mathbf{K}_{\text{pre}}$

can be mapped to an interpreted trace $\mathbf{K}_\mathrm{s}$ which is done in the steps described in algorithm lines 4 - 6. This includes the following.

1. The extraction instructions in U_comb are joined on $(m_\mathrm{id}, b_\mathrm{id})$, with all its according raw messages in $\mathbf{K}_\mathrm{pre}$. This results in a table $\mathbf{K}_\mathrm{join}$ that contains entries of type $(t, l, m_\mathrm{info}, s_\mathrm{id}^\mathrm{rel}, b_\mathrm{id}, m_\mathrm{id}, u_\mathrm{info})$.
2. With this, all u_info of all TVs that are to be extracted are contained with all entries of messages that contain the uninterpreted value of that TV $\mathbf{K}_\mathrm{pre}$. Thus, now an efficient row-wise translation of each row can be performed to extract the value of each TV at each row by using the rules u specified in u_info. For this, u_info needs to contain the byte positions at which the TV value v can be found, the payload l with the content and the translation rule that describes how it maps to the value v. Such evaluation information in particular requires to contain meta information including data types, coding, protocol based fields and translation rules (e.g. Hex to categorical value).

This is exemplified in Table 4.1. As shown there, first, the relevant payload bytes l_rel are extracted row-wise in $\mathbf{K}_\mathrm{join}$ using $u_1 : (l, u_\mathrm{info}) \mapsto l_\mathrm{rel}$ which gives $\mathbf{K}_\mathrm{join2}$. With this, the interpretation information is used to extract the TV values with $u_2 : (l_\mathrm{rel}, m_\mathrm{info}, u_\mathrm{info}) \mapsto k_s = (t, s) = (t, (v, s_\mathrm{id}))$ resulting in the interpreted trace in $\mathbf{K}_\mathrm{s}$. In the proposed DM pipeline a large amount of traces is handled, which requires to extract relevant dimensions on a large scale. The approach is well suited for that purpose, as it allows to process large traces by providing tabular operations that let it be distributed in Big Data frameworks, which allow for automated and systematic extraction on a regular basis. Scalability is enhanced further by the memory efficiency of that approach. That is, traces are stored in raw format $\mathbf{K}_\mathrm{b}$ which is more efficient than translating all $\mathbf{K}_\mathrm{b}$ to $\mathbf{K}_\mathrm{s}$. For instance considering five messages with 10 bytes per message where per message only one byte is relevant for TV extraction, would result in a $\mathbf{K}_\mathrm{s}$ of 10 times the size as proposed here. Moreover other existing tools [96] need to ingest and cache traces of multiple journeys under inspection per analysis. However, once parameterized, the preprocessing allows to automatically extract a trace of relevant TVs at this stage. This allows the consequent semi-automated Specification Mining approach to start directly from the resulting trace, without requiring prior expert intervention during preprocessing every time that a new trace is ingested.

4.2.4 Sequence Reduction

After interpreting the data the method uses a defined reduction technique and type-dependent processing of TVs to achieve a representation that captures the main essence of the data. For this the data is reduced in two ways. First, extensions might be added to the data. That is, by processing the states of TVs further meta information can be extracted, which would be represented as a separate TV. This step is not required for Specification Mining, as the goal is to capture the behavior of the system as is, rather than in terms of meta information. Second, reduction of the data such that only the essence of each TV remains, which is especially useful for Specification Mining as redundancies lead to complex models and thus, overly complex specifications.

numeric

t	ŝ	b_{id}
2s	(45°, wpos)	FC
2.5s	(60°, wpos)	FC
...	...	...

ordinal

t	ŝ	b_{id}
2.1s	(high, heat)	K-LIN
2.7s	(medium, heat)	K-LIN
...	...	...

nominal

t	ŝ	b_{id}
1.0s	(driving, state)	DC
50.1s	(parking, state)	DC
...	...	...

binary

t	ŝ	b_{id}
1.4s	(ON, belt)	FC
22.2s	(OFF, belt)	FC
...	...	...

Figure 4.4: Example of four $\mathbf{K}_{\mathrm{red}}$ each consisting of TV instances of one TV type with four different data types. Those TVs need to be processed based on their data type [4].

The method provides a parameterizable, tabular approach that allows to perform those steps in a distributed and automated manner. Further, at this stage the heterogeneity of the trace is resolved, by performing type-dependent processing that results in a homogeneous data format, with this allowing to perform consequent semi-automated Specification Mining.

4.2.4.1 Reduction

Multiple approaches for reduction of traces towards specific Data Mining tasks were introduced, which are mostly instance [97] or feature [98] selection algorithms. Reduction of traces in the context of software and distributed networks includes reduction of repeated data points [99], pattern-based approaches to map multiple trace segments on a representative [100], sampling techniques [101], clustering approaches to select representative clusters for traces [102] and compression [103]. Those are either restricted to certain system types or tailored to a certain task. However, for a general task-specific analysis of in-vehicle traces, a minimal parameterization for reduction is required. Therefore, unlike in those approaches, in the approach of this chapter database operations are used for reduction, which are performed according to a condition set that is specified by the domain-expert. Also, traces of large distributed systems contain special characteristics that can be well exploited for condition-based reduction. Those include defined communication patterns and channels, routing of identical TV instances on multiple channels, open or closed information flows, sending conditions, sender and receiver information or channel specific information. Such information is well-known and thus, allows for a defined extraction of relevant parts of a trace. For instance, when considering specifications that concern behavior of specific components, only TV instances of defined nodes in the distributed system could be extracted. This reduction is performed in two steps, that include TV splitting with successive type-dependent processing and using constraints. Further, the extension of the trace is performed as separate step. This step is included here for the sake of completeness while it is excluded in the proposed DM pipeline. Those three steps are described in the following.

TV Splitting: The data type of a TV determines how it is processed. This is why in, line 8 the trace $\mathbf{K}_\mathrm{s}$ is split according to its data type. For this the TV types $\Sigma^* = \{s_1^*, s_2^*, ..., s_r^*\}$ that remain after the previous filtering steps are considered, where $\Sigma^* \subseteq \Sigma$. Thus, the trace is considered in terms of each TV individually, i.e. a sequence of the trace that is filtered by a certain TV which is defined as $\mathbf{K}_\mathrm{s}^{s_i^* \mathrm{id}}$. For instance this could be the TV MES that only contains the position of the wiper over time, which is declared in that sense as $\mathbf{K}_\mathrm{s}^{s_\mathrm{id}=wpos}$. This is exemplified in Figure 4.2.
Next, in traces recorded on such systems identical TV instances might have been sent and thus, recorded multiple times. As such entries contain the same information the Specification Mining approach might learn correlations between those if kept in the data set. Further, any redundant information increases computational costs. That is why, TV instances of one of those entities is processed only. For this the algorithm checks equality of TVs in line 9 by doing

$$e : \mathbf{K}_\mathrm{s}^{s_i^* \mathrm{id}} \mapsto (\mathbf{K}_{s_\mathrm{rep}}^{s_i^* \mathrm{id}}, \mathbf{K}_{s_\mathrm{cor}}^{s_i^* \mathrm{id}}) \tag{4.5}$$

This gives a representative sequence $\mathbf{K}_{s_\mathrm{rep}}^{s_i^* \mathrm{id}}$ and a set $\mathbf{K}_{s_\mathrm{cor}}^{s_i^* \mathrm{id}}$ of the according sequences $\mathbf{K}_{s_\mathrm{cor}}^{s_i^* \mathrm{id}}$. In the following the naming of this sequences is declared as $\mathbf{K}_\mathrm{sep} := \mathbf{K}_{s_\mathrm{rep}}^{s_i^* \mathrm{id}}$.

Constraint Reduction: The expert may specify a set of constraints in order to reduce the trace systematically and specific to his data. Therefore, in this step elements of the constraint set can be marked and filtered for relevant elements of $\mathbf{K}_\mathrm{sep}$. For instance, for the task of Specification Mining at this stage communication information might be excluded. That is, as a TV might contain information about errors during transmission and thus, may contain values such as *bad state* or *missing TV*. Such constraints are defined as a set $C = \{c_i | 0 < i < m, i, m \in \mathbb{N}\}$ with elements $c = (s_\mathrm{id}, d, F)$, where s_id defines the TV for which c is to be applied. As a result, if d is true, all functions $f \in F$ are applied, where f can be a row-wise or an aggregation operation which are inherently distributable in Big Data systems. Such functions might be filtering conditions, such as the computation of the temporal gaps between subsequent rows using f. If that gap is acceptable it the row will evaluate to *true* and will be *false* otherwise. Programmatically those constraints are applied in line 10 by joining C with $\mathbf{K}_\mathrm{sep}$ on their TV type, which results in a reduced trace $\mathbf{K}_\mathrm{cond}$. With this, $f \in F$ is computed if d holds yielding a value e as follows:

$$f : k_\mathrm{cond}.e = \begin{cases} true & \text{if } \exists f_i \in F, \text{ with } f_i(k_\mathrm{sep}) = true \\ false & \text{otherwise} \end{cases} \tag{4.6}$$

If any f is true, so is e. Thus, in line 11, $\mathbf{K}_\mathrm{red}$ is found by filtering for f being false leaving only elements that are potentially relevant to Specification Mining at this stage. In particular this is, that only TVs with values that contain state information (**R**, **W** or **Q**) are remaining.

Extension Rules: Extension rules are used to extract further meta-information that is included in the trace. Such information can be declared as separate TV and systematically extracted from the trace. Such TVs are declared as $W = \{w_1, w_2, ...\}$ when added to the trace. The meta information is obtained by applying defined functions on $\mathbf{K}_{\text{red}}$, which is described in line 12. Instances that are computed are added to the trace as new instances of TVs $\hat{w}$. Those are defined similar to TV types as $w = (v, w_{\text{id}})$ with v as value and w_{id} as an identifier associating w to its corresponding TV type. An instance of this TV type is denoted as $\hat{w}$. An example of such meta-information is shown in Table 4.2, where the gap between two consecutive instances of the TV wpos are considered as meta information. This type of rules is not of relevance in the Specification Mining pipeline that is introduced in this thesis.

t	$\hat{\mathbf{w}} = wposGap$	$\mathbf{b}_{\mathbf{id}}$
2s	(0.5, wposGap)	FC
2.5s	(0.4, wposGap)	FC
2.9s	(0.45, wposGap)	FC
...	...	...

Table 4.2: Extension: Gap between wpos TVs from sequence $\mathbf{K}_{\text{s}}^{s_{\text{id}}=wpos}$.

4.2.4.2 Type-Dependent Processing

Next, TVs are processed depending on their type. The procedure for homogenization is required in Specification Mining to allow for semi-automated preprocessing that directly results from this step.
TV types of $\mathbf{K}_{\text{red}}$ are diverse in data type. However, it is cost-efficient for further analyses to have data available homogeneously in a common representation, as then, inspection can be performed instantly. Each interpreted sequence $\mathbf{K}_{\text{red}}$ is either nominal, ordinal, binary, numeric or of mixed type. This includes sequences, where numeric values describing a system property (e.g. velocity of a car) are mixed with network specifics (e.g. TV invalid). In those cases, it is not enough to filter out nominal data, as it could resemble a fault occurrence. This also holds for outlier handling for similar reasons. E.g. outliers need to be removed during symbolization, while being in the data set after the processing procedure.
For homogenization, TV types are grouped in three classes that require different processing approaches corresponding to criteria that were determined through inspection of more than 1000 TV types. Those are presented in the following.

Data Types - Criteria: Each TV instance sequence needs to be classified in advance. Therefore criteria are defined.

$$Z = (z_{\text{type}}, z_{\text{rate}}, z_{\text{num}}, z_{\text{val}}). \tag{4.7}$$

First, the data type

$$z_{\text{type}} \in \{S, N\} \tag{4.8}$$

is either a String S or a Numeric N. The affiliation

$$z_{\text{aff}} \in \{F, V\} \tag{4.9}$$

is used to distinguish between values that express a functional property F and the ones that define validity V of either the sent message (e.g. message invalid), the TV instance itself (e.g. TV invalid) or a functional component (e.g. object invalid). For numeric values

$$z_{\text{rate}} = \begin{cases} H & \text{if } \frac{n}{\Delta t} > T \\ L & \text{otherwise} \end{cases} \tag{4.10}$$

is used to differentiate between values changing at a high rate H, such as sensor TV types and those changing at a slow rate L, using a threshold T, that is determined with domain knowledge. z_{rate} is the number of values n in active segments of duration Δt. Furthermore, the number of different values z_{num} of a functional property information is used. For ordinal values it needs to be specified if TV values of a type contain a comparable valence:

$$z_{\text{val}} \in \{true, false\};\ z_{\text{num}} \in \mathbb{N} \tag{4.11}$$

According to Z, each TV instance sequence $\mathbf{K}_{\text{red}}$ is assigned a processing branch according to the mapping in Table 4.3.

Branch α: In general $\mathbf{K}_{\text{red}}$ contains at this point both elements that are numerical and nominal. This is because, when transmitted, data is a value, erroneous or given a meaning. Nevertheless, those nominal values are impeding the processing of numeric values, e.g., during trend estimation. Thus, $\mathbf{K}_{\text{red}}$ is split in a sequence $\mathbf{K}_{\text{nom}}$ of nominal and a sequence $\mathbf{K}_{\text{num}}$ of numerical elements, which are processed separately as follows:

$$\mathbf{K}_{\text{red}} \mapsto (\mathbf{K}_{\text{num}}, \mathbf{K}_{\text{nom}}) \tag{4.12}$$

For $\mathbf{K}_{\text{num}}$ the data is split in a trend and range part as introduced in [104]. To exclude outliers from this estimation, first a filtering for outliers is performed by windowing the data and per window, removing all values that are further than three times the standard deviation from the mean value. This results in $\mathbf{K}_{\text{num}}$. However, as outliers are potentially meaningful, those elements are marked and stored in $\mathbf{K}_{\text{num}_{\text{out}}}$. The mapping can be written as

$$\mathbf{K}_{\text{num}} \mapsto (\mathbf{K}_{\text{num}_{\text{out}}}, \mathbf{K}_{\text{num}_{\text{rep}}}) \tag{4.13}$$

Next, for symbolization the mapping m on $\mathbf{K}_{\text{num}_{\text{rep}}}$ is used. This mapping applies exponential moving average smoothing to remove noise, uses SWAB [95] and linear regression to determine the trend and SAX [88] for symbolic quantification. This, results in a nominal tuple of trend and range for each element giving $\mathbf{K}_{\text{num}_{\text{clean}}}$ by using

$$m : \mathbf{K}_{\text{num}_{\text{rep}}} \mapsto \mathbf{K}_{\text{num}_{\text{clean}}}. \tag{4.14}$$

To return outlier and nominal information, $\mathbf{K}_{\text{num}_{\text{clean}}}$ is merged with $\mathbf{K}_{\text{num}_{\text{out}}}$ and $\mathbf{K}_{\text{nom}}$ giving the sequence

$$\mathbf{K}_{\alpha} = \mathbf{K}_{\text{num}_{\text{clean}}} \cup \mathbf{K}_{\text{num}_{\text{out}}} \cup \mathbf{K}_{\text{nom}}. \tag{4.15}$$

, which is excluded from the DM pipeline that is propose here.

Branch β: For the same reasons, a similar approach is used for ordinal values. Those are split by applying condition z_{aff} on each element in $\mathbf{K}_{\text{red}}$ giving a nominal part $\mathbf{K}_{\text{V}}$

z_{type}	z_{rate}	z_{num}	z_{val}	Data Type	Processing Branch
N	H	> 2	$true$	numeric	α
N	L	> 2	$true$	ordinal	β
S	$H \cup L$	> 2	$true$	ordinal	β
S	$H \cup L$	$= 2$	$true$	binary	γ
S	$H \cup L$	> 2	$false$	nominal	γ
N	$H \cup L$	$= 2$	$true$	binary	γ

Table 4.3: Map TV instance sequences to data type and processing branch [4].

where V holds and a functional part $\mathbf{K}_{\text{F}}$ where F holds. This gives

$$\mathbf{K}_{\text{red}} \mapsto (\mathbf{K}_{\text{F}}, \mathbf{K}_{\text{V}}). \tag{4.16}$$

Similar to α, $\mathbf{K}_{\text{F}}$ is first translated into a numerical equivalent, analyzed for outliers and the trend is determined using the gradient. This results in a tuple of gradient as trend and the value of the element, giving sequence $\mathbf{K}_{\text{F}_{\text{clean}}}$. Outliers are again stored to sequence $\mathbf{K}_{\text{F}_{\text{out}}}$ and can optionally be merged (which is not done in Specification Mining), giving

$$\mathbf{K}_{\beta} = \mathbf{K}_{\text{F}_{\text{out}}} \cup \mathbf{K}_{\text{F}_{\text{clean}}} \cup \mathbf{K}_{\text{V}}. \tag{4.17}$$

Branch γ: If a low amount of values is given no transformation is needed and all values are treated as nominal values. Splitting similar to β in $\mathbf{K}_{\text{V}}$ and $\mathbf{K}_{\text{F}}$ is possible. Also, similar to B, outliers can be removed optionally. This results in $\mathbf{K}_{\gamma}$.
Notably, for the Specification Mining task that is following this procedure, outliers are removed, as the aim is to reduce noise and to filter for the essence of the true behavior of each domain-specific function and its functional procedures.

4.2.5 Final Representation

To obtain the final representation the resulting sub sequences of $\mathbf{K}_{\alpha}$, $\mathbf{K}_{\beta}$ and $\mathbf{K}_{\gamma}$ are merged, which yields a homogeneous representation of the trace $\mathbf{K}_{\text{rep}}$.

State Representation: Multiple representations can be extracted from $\mathbf{K}_{\text{rep}}$. For instance, when vectorizing the state of the system for Machine Learning algorithms, such as Recurrent Neural networks a good representation is to store all states at each time step, while recording a time step each time a value changed. An example of this is shown in Table 4.4.
Notably the resulting format decodes a MSS, e.g. if one column is considered only and duplicates are removed a state sequence per TV is extracted. MSSs are used as an input to the proposed Specification Mining approach. For this, it suffices to store the changing TV only resulting in a format as shown in Table 4.5.

4.3 Discussion

For Specification Mining the result of $\mathbf{K}_s$ is further processed, which is extracted after the Interpretation phase. Further, this preprocessing produces the MSS $\mathbf{K}_{\text{rep}}$, which is referred to as $\mathbf{K}_n$ in the remainder of this work. The latter, is produced by removing

t	$s_{\text{headlight}}$	$s_{\text{levercontrol}}$	s_{speed}	$s_{\text{indicatorlight}}$	$s_{\text{lightswitch}}$
2	off	default	(high,increasing)	off	default
4	off	**pushed up**	(high,increasing)	off	default
4.25	off	pushed up	(high,increasing)	**left on**	default
7	off	**default**	(high,increasing)	left on	default
7.22	off	default	(high,increasing)	**off**	default
14	off	default	**(high,steady)**	off	default
20	off	default	(high,steady)	off	**turned halfway**
20.1	**park light on**	default	(high,steady)	off	turned halfway
22	park light on	default	**outlier** $v = 800$	off	turned halfway
23	park light on	default	(high,steady)	off	**turned full**
23.5	**headlight on**	default	(high,steady)	off	turned full

Table 4.4: Exemplary state representation of TV instances of the function lights combined with driving speed [4].

t	s_{id}	v
2	$s_{\text{headlight}}$	off
4	$s_{\text{levercontrol}}$	pushed up
4.25	$s_{\text{indicatorlight}}$	left on
7	$s_{\text{levercontrol}}$	default
7.22	$s_{\text{indicatorlight}}$	off
14	s_{speed}	(high,steady)
20	$s_{\text{lightswitch}}$	turned halfway
20.1	$s_{\text{headlight}}$	park light on
23	$s_{\text{lightswitch}}$	turned full
23.5	$s_{\text{headlight}}$	headlight on

Table 4.5: Exemplary state representation of TV instances of the function lights combined with driving speed.

successive identical states. Once parameterized, those representations are automatically generated and are used for the consequent steps of the proposed pipeline.
The presented approach is performed in a semi-automated manner and per domain. This is well suited here, as the expert is usually responsible for a certain function, that consists of a set of TVs that are relevant to it. Thus, at this stage an initial set of TVs is specified in U_{rel}. While the expert does not exactly know all relevant TVs at this stage, he is able to provide a broad set of TVs which he expects to potentially influence the inspected functional procedures. Additionally, reduction constraints C can be added here by the expert. For Specification Mining this might be the removal of communication information, e.g. the filtering for state information only.
Advantages of this approach include, that once parameterized, the resulting trace can automatically be extracted on each ingested instance of a system. Then, from there each expert can proceed with the further steps of the pipeline. This is further enhanced by the fact that the trace can be reduced significantly towards the analyzing domain, which will be shown in the evaluation section. This allows to save both computational as well as memory costs. Moreover, this preprocessing results in a noise reduced and homogeneous trace.
Due to those reasons this step is important to allow for a systematic and automated mining of Specifications at a large-scale.

4.4 Evaluation of Framework Performance

Within the proposed pipeline the aim is to reduce the size of any incoming trace of a massive data set such that the resulting format can be used by the domain specialist for further Specification Mining. Two particularly important properties for this include the time required for the transformation to take place and the reduction rate that is achieved through this step. The time aspect was covered in comparison to existing methods and is shortly revisited here. Next, to this the reduction rate of the approach is discussed on the same data. This evaluation was first published in [4] and is revised here.

Setup: The evaluation is performed on lines 3 to 11 of Algorithm 4 using Apache Spark on a cluster with 70 servers, Infiniband QSFP, 20 virtual CPUs, per node two Intel ® Xeon ® processors E5-2680v2 with 256 GB DDR3 RAM using 5 virtual CPUs and 10 servers with 3 GB RAM per executor and 4 GB RAM per driver node. The remaining part of the algorithm is not considered, as methods used there are evaluated in respective literature, e.g., SWAB Algorithm [95]. In the given data sets TV instances are sent with constant cycle times and often, for subsystem inspection, only changes of TV values are relevant. Thus, as constraint reduction the choice was to remove TV instances where values did not change over time.

Data: Three representative data sets are inspected, which were recorded from one modern premium vehicle during 20 hours of driving. Their statistics are described in Table 4.6. From this trace, per data set only message instances are extracted that contain any of the TV types of the chosen data set. Thus, all message instances in $\mathbf{K}_{\mathrm{b}}$ contain at least one TV instance of the corresponding data set. The first data set is the Lights data set *LIG* with trace $\mathbf{K}_{\mathrm{b}}^{\mathrm{lig}}$ and relevant TV types $U_{\mathrm{rel}}^{\mathrm{lig}}$ containing TV types

	SYN	LIG	STA
# TV types	13	180	78
# TV types - α	6	27	6
# TV types - β	4	71	1
# TV types - γ	3	82.	71
# TV types	13,197,983	12,306,327	4,807,891
∅ TVs per message	1,47	5.11	3.66

Table 4.6: Statistics of the three data sets used here.

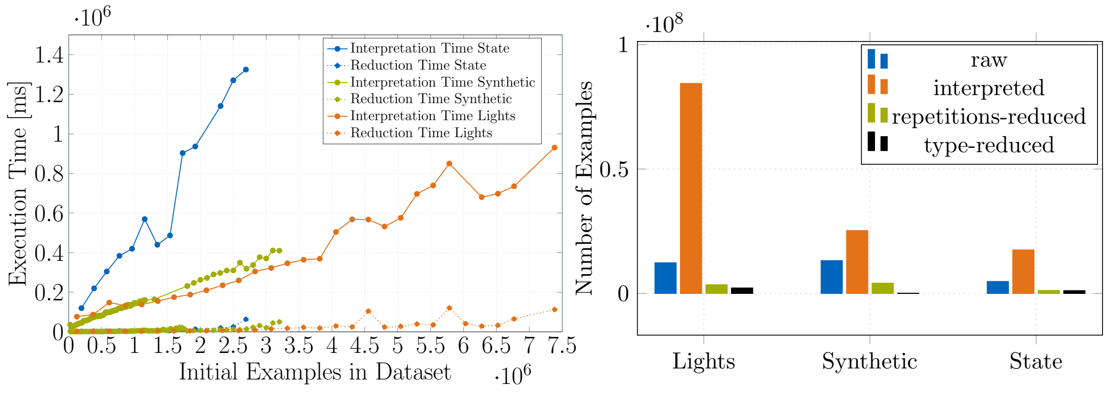
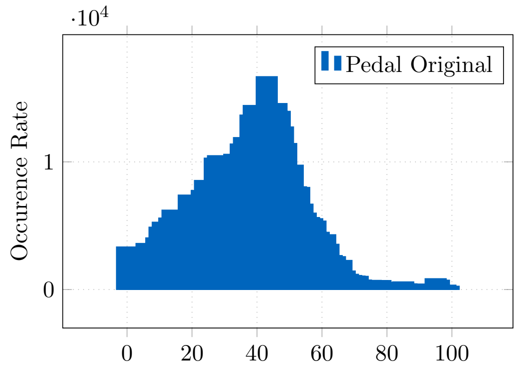


Figure 4.5: On the left the execution time after interpretation and removal of identical consequent TV instances is depicted, when the number of initial examples is varied. The right figure shows the size after each processing step when the whole trace and all TVs are interpreted, reduced and symbolized [4].

exchanged within the light function and include e.g. the brightness or the state of front lights. The State data set *STA* (with $\mathbf{K}_\text{b}^\text{sta}$ and U_rel^sta) has TV types that describe the cars State, e.g. its driving state (parking, driving, repair). The third set is synthetic *SYN* ($\mathbf{K}_\text{b}^\text{syn}$ and U_rel^syn) and generated from 13 representative TV types from data sets of different functions which are replicated 9 times. Per replicated TV type, a minimal time shift is added and a unique m_id is assigned yielding a data set with 130 TV types. The interpretation rules per data set U_rel are stored in a Hive table.

Execution performance: The approach was run with a constant number of TV types and step-wise increase a subset of $\mathbf{K}_\text{b}$. Also, per data set, all TV types are extracted giving one $\mathbf{K}_\text{red}$ per TV type. I.e., in each data set, all entries are interpreted. The results that were presented in [4] are shown in Figure 4.5.
Execution time increases with the number of additional examples in the data set. This growth is linear as interpretation is performed per row, more examples are processed per node and this processing has complexity $O(n)$. Fluctuations result from communication that is performed within the cluster and the distribution of types of TV instances among nodes, as processing differs per data type. It was found that interpretation is expensive in terms of execution time and, thus, early reduction shows to be advantageous here. This approach has reasonable results for *STA* and *LIG*, as e.g. the interpretation of 2.6 million examples was processed in 1324 seconds and 7.4 million examples in 930 seconds.



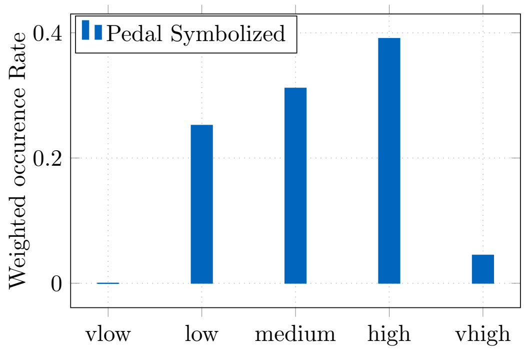


Figure 4.6: On the left side the occurrence rate of values for the original TV values of the angle of the acceleration pedal are shown. The occurrence rate after symbolization is shown on the right.

Journeys	Trace rows $\cdot 10^9$	Extracted rows $\cdot 10^6$	# Extracted TVs	Extraction time Proposed [min]	Extraction time in-house [min]
1	0.481	12.751	9	9.58	41.66
1	0.481	79.466	89	168.05	41.66
7	4.286	94.013	9	62.00	372.88
7	4.286	586.124	89	183.25	372.88
12	5.901	133.619	9	87.62	504.27
12	5.901	833.066	89	269.65	504.27

Table 4.7: TV extraction times for massive traces as introduced in [4].

Here, computational power was restricted to 10 nodes, thus, on a bigger scale an increase of the underlying resources can improve this performance.

Comparison: Further, this approach was compared to an OEM's in-house tool [96] (comparable to Wireshark), that was run on an HP™ Z-840 equipped with two Intel® Xeon® E5-2640 v3 2.60GHz CPUs and 96GB of RAM. The approach of this chapter was run on 10 nodes, 5 cores and 10 GB of RAM per executor and 20 GB of RAM per driver. Here the extraction procedure was performed on massive traces of billions of rows, with results as shown in Table 4.7.
For extraction this the in-house tool does the interpretation on ingest and in [4] the time for ingest was measured as extraction time, while for the preprocessing in the framework the time for interpretation until it is written to the database was measured. The same amount of traces was ingested to both approaches with a fixed number of TVs, which is the second column in Table 4.7. Based on this the execution times in both methods for traces of various sizes was discussed. It was found that for 89 extracted TVs from 12 journeys a reasonable extraction time of 269.65 minutes was needed, that is nearly twice as fast as the existing solution with a time of 504.27 min. For 9 TVs the tool needs 87.62 min for 9 TVs while the existing tool requires 504.27 min which shows a improvement of factor 5.7. The in-house tool loops through the data to extract relevant TVs , which yields a linear scaling with data size. This extraction time does not change with the number of extracted TVs as extraction is done within one loop. This existing solution is sufficient if individual journeys are considered. However, when extracting

TVs from massive traces (i.e. multiple journeys) this becomes inefficient and distributed approaches become essential. Thus, the processing approach discussed in this chapter is well suited to work as a basic preprocessing within the systematic DM pipeline proposed here, as it can reduce the size to a trace relevant to a domain expert within a reasonable time. This suitability is further given as the process is parameterized once and the results are automatically written to a database that is used for the further semi-automated Specification Mining task.

Reduction performance: Just as for execution time the reduction rate r was measured and results are shown in Figures 4.5 (middle). r is measured as number of examples that remain after interpretation n_{proc} (and number of examples after reduction, respectively) and the initial number of examples n_b in each data set using $r = \frac{n_{\text{proc}}}{n_{\text{b}}}$.
In a separate experiment for each data set also type-dependent preprocessing is applied for all extracted TV instances and types. The reduction results are shown in Figure 4.5 (right).
Results show ($r > 1$) that interpretation produces multiple TVs extracted per message, because as Table 4.6 shows on average multiple TV types are present per message. The values in Table 4.6 resemble the average number of TV types in the same message type. For sets *SYN* 1.47, for *LIG* 5.11 and for *STA* 3.66 TVs should be extracted on average. This is true for *SYN*. For the other sets, this value is bigger, as the distribution of occurrence of each message type is unknown. E.g. a message instance with 8 TV types could be dominating. It can be seen that traditional approaches, which extract all available TV types per message, would cause a massive data increase here, while with the given algorithm increase in size due to interpretation is minimized.
After reduction, the percentages of TV instances were reduced to be below 34 %, which shows that for functional analyses only a fraction of data is relevant. For *SYN* only about 23 % ($=\frac{0.34}{1.46}$), for *STA* 6.5% ($=\frac{0.34}{5.2}$) and for *LIG* 3.65% ($=\frac{0.34}{7.3}$) of TV instances are relevant (e.g. 5.2 as mean of interpreted size for *STA*). Similar to the execution time, the curve showing the interpretation size fluctuates as added message instances can contain different numbers of TVs.
Next, type-dependent preprocessing was performed on the whole trace, per TV type, resulting in a symbolic trace. After symbolization, further reduction of data is performed by removing repeating elements of the same content, as symbolization could have led to similar repeating symbols. Results are shown in Figure 4.5 (right). When compared to the number of interpreted values for *SYN*, data size are reduced to 0.35 % , for *LIG* to 2.6 % and for *STA* to 6.4 %. With this reduction performance, visual inspection and further Data Mining become applicable.

Filter evaluation: As an example for the symbolization procedure the reduction quality of the TV type "Acceleration Pedal Angle" is shown with its occurrence distribution before and after symbolization for the whole trace. As the symbolic value is valid until a state change occurred, we, per state, aggregate the relative time the Pedal angle had a certain symbolized state and binned it. As original values are sent with high cycle times, those were binned in 100 bins of equal range. Figure 4.6 shows that for the pedal TV type symbolization yields a good approximation and that SAX is a valid approach for symbolization as it keeps approximate characteristic while reducing the data.

4.5 Summary and Conclusion

In this chapter the focus was on preprocessing of raw communication traces in the context of automated Specification Mining. For this, we revised the approach that was presented in [4], which is used at the first step of the proposed DM pipeline. This approach is a fully parallelisable preprocessing framework that allows for systematic and efficient interpretation, reduction and homogenization of communication traces. Using its parametrization in terms of constraints, experts are able to specify both TVs and constraints for automated extraction, which ultimately yield a domain-specific representation of the data that is used as an input for the further steps of the proposed pipeline. The evaluation presented here and in [4] showed on three real world data sets, that this framework is reasonable in run time, has low loss of information and can automatically extract relevant aspects of the trace.
Here, the method is used as preprocessing for the scalable mining of specifications. Nevertheless, this approach has great potential to be extended towards further fault inspection tasks. Thus, in the future the framework can be extended by systematic anomaly detection and ranking anomalies in terms of error potential. This information could then, be automatically transformed in extension rules w, that detect similar anomalies in further runs. Another interesting aspect would be to do this autonomously by detecting faults, learning bad behavior and by using this to generate new reduction and extension rules or to associate relevant meta-data. Moreover, for detected errors, data can be enriched with counteractions that were undertaken. Using those as target variables for classification, models for automated recommendations of counteractions can be generated.

5 Clustering High-Dimensional Sequences

At this stage it is assumed that a preprocessed MSS $\mathbf{K}_n$ and MES $\mathbf{K}_s$ was extracted that represents the overall trace. This trace consists of multiple data points of an object over a long period of time. In the second step of the proposed pipeline this trace is used to identify correlating TVs. Starting from the MES $\mathbf{K}_s$ a clustering approach C with hyper parameters P_C is used to assign each TV S_i a cluster k using $a(S_i)$, where $k \in \{1, ..., K\}$.

$$a = sigcl(\mathbf{K}_s, C, P_C) \tag{5.1}$$

Resulting from this, K clusters of TVs are found, which results in sub traces $\mathbf{K}_s^k$ that only contain TVs $\hat{S}$ that were chosen by the expert and that are part of a certain cluster assignment $\hat{a}$. This gives the output

$$\mathbf{K}_s^k = sel(\mathbf{K}_s, \hat{S}, \hat{a}), \text{ where all } S_i \in \hat{S} \text{ and } a(S_i) = \hat{a} \tag{5.2}$$

$$\mathbf{K}_n^k = sel(\mathbf{K}_n, \hat{S}, \hat{a}), \text{ where all } S_i \in \hat{S} \text{ and } a(S_i) = \hat{a} \tag{5.3}$$

This step is essential, as the resulting MES $\mathbf{K}_s$ is of high dimension, while only a subset of TVs are relevant to the functional procedures of the analyzing domain, e.g. for the analysis of the active cruise control of a car TVs such as the state of the wiper are irrelevant. Further, both to identify all relevant dominant behaviors and specifications in a trace, complexity is significantly reduced by considering subsets of TVs for this process only.

In complex distributed systems the identification of TV groups is in general not possible to be done manually, as dimensionality is too high, the integrated system is developed by multiple domains, the system is incrementally evolving throughout the development process, functionality of the system is growing, multiple disjunct functional procedures might be present in the trace and a high degree of experience is required. Additionally, the massive number and high complexity of TVs, that results from heterogeneous data formats across multiple functions, aggravates this process. It becomes increasingly difficult to identify TVs that belong together and need to be considered for domain-specific Specification Mining. Thus, by identifying correlating TVs automatically and refining this selection with expert input, functional procedures are found on relevant subsets of TVs only. This reduces both complexity of the consequent mining approach, as well as it increases the relevance of the found functional procedure towards the analyzing domain.

Thus, in this chapter a method for *sigcl* is presented. For this a feature-based clustering approach is combined with expert input on a trace $\mathbf{K}_s$ for grouping of correlated TVs. Relevant features are determined and multiple clustering approaches C are compared.

Therefore, it is assumed that, TVs that change within similar periods of time are highly likely to be part of a common functional procedure or at least a common function. The more frequently a subset of TVs occurs together, the more likely it is that this occurrence was not random, but due to correlation.
Moreover, expert input needs to be included, as this grouping may possibly not include all relevant TVs. Oftentimes a subset of TVs remains in a steady state for a long time. The value of this state is crucial to the respective functional procedure or if the procedure occurs at all, e.g. being in the driving state as a car, it is more likely to observe a braking procedure than in the parking state. Further, it might be the case, that separate clusters of TVs are found that correspond to the same functional procedure, where e.g. one cluster corresponds to the left and the other cluster to the right part of a procedure. By using a simplified and grouped representation of TVs, that is extracted during this step, experts are able to include relevant TVs in the further process. Above that, granularity of clustering, i.e. the hyper parameters of the clustering approaches, are hard to assess automatically as the highest degree of similarity does not always correspond to common functional procedures. Thus, at this step expert input is used in the proposed semi-automated clustering approach.
A further challenge results from the high dimensionality, i.e. high number of samples, of each TV. Due to the *curse of dimensionality* clustering gets increasingly imprecise as the pairwise distance between vectors increases with the number of dimensions. [105].

Chapter Outline: First, related works in the field of temporal clustering are presented in Section 5.1 before the clustering procedure of [5] is presented in Section 5.2. Lastly, in Sections 5.3 and 5.4 the evaluation of this procedure is discussed. Notably, this chapter is based on the paper presented in [5]. The findings of this work are used as part of the proposed DM pipeline.

5.1 Related Works

Related Works include temporal clustering approaches that were presented in the past, as well as its applications.

5.1.1 Temporal Clustering Approaches

Temporal Clustering can be mainly grouped in approaches working on raw data, classical feature-based approaches and novel clustering approaches that originate from the field Representation Learning. All these approaches often use the same clustering approaches, while the main difference lies in the type of data that is processed and the way of representing the input within these approaches.

Raw Data: This type of approaches is applied on raw time-series, which are clustered according to its characteristics. Based on these various clustering approaches, such as K-Means can be used [106]. For that, often a distance function is defined, which describes the similarity between corresponding TVs. This includes classical distance measures such as Euclidean or Mahalanobis distance, as well as measures such as Dynamic-Time Warping [107]. Further, for short time-series the glsSTS - Distance was introduced in combination with a variation of the K-Means approach [108]. However, these works focus on short time series, while these are not suited for longer sequences. In this chapter the focus is on clustering of large sequences, which makes these approaches not applicable here.

Feature-based: These methods extract features that represent characteristics of a time-series or sequence and groups sequences according to these. This, is especially useful for long sequences, as it allows to break down the sequence to its essence, which enables a more computationally efficient processing. These approaches mainly differ in the type of features that are chosen and the representation of the sequence. Further, such approaches are less prone to noise as it is usually filtered out during preprocessing and has less influence within the approach [109].
In [109] features were extracted per TV from control flow information using static features including mean, standard deviation or skew. Here, a neural network was used for clustering. In [110] time-series are clustered using the trend, seasonality, periodicity or skew using Self Organizing Maps (SOM)s. The authors of [111] used Wavelet coefficients, that, depending on their level, are able to capture both high frequent characteristic and coarse tendencies of a sequence. In [112], the sum of differences between consecutive elements is used.
For longer sequences it is useful to segment the overall trace and to extract features within such segments, which can be done bottom-up, top-down or using sliding window approaches [113]. This is for instance done in [114], where TVs are windowed and per interval relevant features are computed, which is similar to the approach applied in this chapter. Also, in [115] a time series of the electrical usage is segmented and features extracted, that include variance, skew of energy.
Main advantages of such approaches include the possible reduction of dimensionality, robustness against outliers and the ability to process time-series of different lengths. In [116] Piece wise Linear Representation is used for that purpose, which uses a bottom up approach that merges segments while keeping an approximation error per unit low.
Static window sizes might however, simplify away relevant characteristics for the case of patterns of varying lengths or for patterns that are part of multiple windows, [117]. This is be solved by performing segmentation not statically but rather at Perceptually Important Points [118], which cut sequences at points apart of trends, or by slicing windows such that entropy is minimized [119]. Overlapping windows were used in [120] with static features. Such approaches are mostly limited to time-series rather than MES of heterogeneous data types as it is the case here.
Feature-based clustering approaches are mainly grouped in partitioning algorithms, such as K-Means [121] or K-Medoids [122], hierarchical, such as agglomerative [123], density based, such as DBSCAN [124], raster based, such as STING [125] or other approaches such as Affinity Propagation [126] and Self Organizing Maps [127]. These approaches will be discussed in Section 5.2.1.

Representation Learning: In representation learning features are not extracted manually, but rather the approach is able to learn relevant features automatically. In recent years this is mostly achieved by using Deep Learning approaches. Such approaches include general clustering approaches and methods specified on temporal clustering.

According to [128] general clustering approaches are categorized according to its network architecture to the following groups. First, this is Auto encoders that are trained to reconstruct the original data, allowing these to represent its inherent features. This includes algorithms such as Deep Clustering Networks [129], Deep Embedding Network [130] or Deep Continuous Clustering [131]. Second, these are methods based on feed-forward networks that are trained by specific clustering loss. Approaches for this are Deep Embedded Clustering [132], Discriminatively Boosted Clustering [133] or Deep Adaptive Image Clustering [134]. Third, approaches based on Generative Adversial Networks which includes Categorical GAN [135] or Information Maximizing GAN [136] and fourth, methods that use Variational Auto encoders, such as Variational Deep Embedding [137], are used in this context.
Further, approaches that are specific to time series data introduced. This includes the approach described in [138], which is called Deep Temporal Clustering, that uses a temporal Auto encoder that is composed of convolutional and bi-directional long short-term memory neural networks. This approach is mainly used on univariate time series. This type of clustering showed to outperform feature-based approaches in terms of accuracy and representational capacity. However, it is computationally expensive, as training deep neural networks requires multiple iterative optimization steps.

5.1.2 Temporal Clustering Applications

Applications of clustering on temporal data are manifold, some of which are the following. K-Means was used in [115] to group power usage based on the Davies-Bouldin-Index. Shaw and King grouped speed measurements recorded in a wind channel [139] using agglomerative clustering. Both approaches were used in [140] for stock market prediction, in [141] to group functional magnetic resonance tomography measurements. SOMs were used in [142] for clustering of power consumption.
In fault diagnosis in [84] features are extracted from multiple signals in order to classify them as normal or abnormal. Also, in [85] the focus is on finding causal relations between individual features of signals and fault types. Grouping of signals was performed in [78], where supervised learning approaches were used to classify signals as internal (state of vehicle) and external (state of environment). However, for massive numbers of signals a supervised approach requires high labeling costs. Many Data Mining approaches where applied to in-vehicle signals, most of which are focused on diagnostics. In [143] diagnostic neural networks are trained for fault classification and in [144] induction motor drive faults are detected using recurrent dynamic neural networks. More recently diagnosis in in-vehicle signals was done for anomaly detection, e.g. by using condition indicators [145]. CAN signals were used for predictive maintenance [12]. In [89] vehicle signals are used to predict compressor faults, and in [146] to model the remaining useful life time of batteries in trucks. Moreover, in-vehicle signals were used in applications, such as detection of scenarios [147] or driver workload monitoring [148].

In this chapter the focus is on feature-based approaches, combined with overlapping windowing and static features per window. Such methods are most promising to effectively represent and group long sequences.

5.2 Feature-based Clustering Approaches

In general, clustering of temporal data is done either by variable or by sample. The further refers to grouping TVs while the latter refers to grouping events of TVs. In this chapter the focus is on grouping TVs that change at time steps of common neighborhood [149] and present an approach to enable this.
This is done by reducing data to relevant features and transforming it to a more expressive space. Further, the influence of individual steps on this approach is discussed, which includes the window size, the selected features and the clustering approach chosen. Notably, the focus is on temporal data that is extracted from large-scale distributed systems, which is characterized by multiple functional procedures that occur jointly with common TVs, by its heterogeneous data types and by its massive size.

5.2.1 Background on Clustering Approaches

As described in Section 5.1 the focus is on feature-based clustering approaches, as these are computationally well suited to represent TVs of large size. Existing approaches are the following which are categorized as proposed in [150].
Formally, clustering refers to the task of separating a data set $|X| = n$ into $k \leq n$ partitions, i.e. in the current context it refers to the partitioning of n TVs into k partitions, where

$$C = C_1, ..., C_k; \bigcup_{i=1}^{k} C_i = X, \tag{5.4}$$

$$\text{with } C_i \cap C_j = \emptyset \text{ for } 1 \leq i, j \leq k \text{ [123]} \tag{5.5}$$

5.2.1.1 Common Distance Metrics

Different distance metrics are used for this clustering most common of which are the following.

- **Minkowski - Norm:** This norm is defined by a constant α for a p-dimensional distance $\mathbf{x}$ between two points, where $\mathbf{x}^{(j)}$ refers to the j-th element in $\mathbf{x}$. That is

$$\|\mathbf{x}\|_\alpha = \sqrt[\alpha]{\sum_{j=1}^{p} |\mathbf{x}^{(j)}|^\alpha} \tag{5.6}$$

- **Manhattan - Distance:** This distance is a Minkowki - Norm with $\alpha = 1$, which is

$$\|\mathbf{x}\|_\alpha = \sqrt{\sum_{j=1}^{p} |\mathbf{x}^{(j)}|} \tag{5.7}$$

- **Euclidean - Distance:** This distance is a Minkowki - Norm with $\alpha = 2$, which is

$$\|\mathbf{x}\|_\alpha = \sqrt[2]{\sum_{j=1}^{p} |\mathbf{x}^{(j)}|^2} \tag{5.8}$$

- **Chebyshev - Distance:** This distance is a generalization of Minkowki - Norm with $\alpha \mapsto \infty$, which is

$$\|\mathbf{x}\|_\infty = \max_{j=1,\dots,p} \mathbf{x}^{(j)} \tag{5.9}$$

5.2.1.2 Clustering Approaches

Partitioning Algorithms: This type of clustering separates all samples in the data set in a given number k of groups [150]. Most important approaches include k-Means, k-Medoids or Gaussian Mixture Models.

- **k-Means:** In this approach the clustering problem is defined as an optimization problem that aims to minimize the error function

$$\bar{\mathbf{c}}_i = \sum_{C_i \in C} \sum_{\mathbf{x} \in C_i} d(\bar{\mathbf{c}}_i, \mathbf{x}) \tag{5.10}$$

  with cluster c_i, data set X, data set element $\mathbf{x} \in X$ and distance function $d(\mathbf{c}_i, \mathbf{x})$. A common algorithm that solves this optimization problem is Lloyd's algorithm. There each cluster i is defined by a prototype, which is the center of the cluster c_i. Each element of the data set $\mathbf{x} \in X$ is assigned a prototype $\mathbf{c}_i$, that has the closest distance $d(\mathbf{c}_i, \mathbf{x})$. Based on this assignment the cluster center is recomputed at each iteration using

$$\bar{\mathbf{c}}_i = \frac{1}{\|C_i\|} \sum_{\mathbf{x}_k \in C_i} \mathbf{x}_k \tag{5.11}$$

  Following this step the next iteration starts with the reassignment. On appropriate initialization the cluster center converge towards a good separation. *Complexity:* The complexity is $\mathcal{O}(nkt)$ where t is the number of iterations [151]

- **k-Medoids:** This approach is similar to k-means. However, in k-means the computed result might be blurred by outliers that negatively affect the computation of the mean. That is why k-Medoids uses the median as a more robust prototype at each iteration. A common implementation is called the Partitioning Around Medoids, while more scalable implementations are Clustering Large Applications (CLARA) or Clustering Large Applications based on Randomized Search (CLARANS) [150].

- **Gaussian Mixture Models:** This approach models each cluster as a Gaussian distribution that best fits the data in each cluster. For this, it assumes the likelihood $p(C_j|\mathbf{x})$ of a data point $\mathbf{x}$ to be part of cluster C_j and models the overall likelihood of all data points as

$$L(X|C) = \prod_{j=1:n} \sum_{j=1:k} \tau_j p(\mathbf{x}_i|C_j) \tag{5.12}$$

where τ_j is the likelihood of cluster C_j.
Clustering is achieved by fitting all distributions onto the data. Multiple algorithms are used for this purpose. Common approaches include the Expectation Maximization (EM) algorithm or Variational Inference. *Complexity:* The complexity of EM is $\mathcal{O}(nk)$ [152]

Hierarchical Clusterings: These approaches perform a sequential decomposition of objects to form clusters. This can be done bottom-up. That is, starting from fine grained clusters per data point groupings coarser clusters are found by grouping such clusters. Also, this can be done top down, where the opposite is the case, i.e. starting from a coarse clustering finer clusters are successively found. This is usually done based on a distance metric [123] that is computed between clusters, which can be

- **Single Linkage:**

$$d(C_a, C_b) = \min_{x \in C_a, y \in C_b} \|x - y\| \tag{5.13}$$

- **Complete Linkage:**

$$d(C_a, C_b) = \max_{x \in C_a, y \in C_b} \|x - y\| \tag{5.14}$$

- **Average Linkage:**

$$d(C_a, C_b) = \frac{1}{\|C_a\|\|C_b\|} \sum_{x \in C_a, y \in C_b} \|x - y\| \tag{5.15}$$

- **Center distances:**

$$d(C_a, C_b) = \|\frac{1}{\|C_a\|} \sum_{x \in C_a} x - \frac{1}{\|C_b\|} \sum_{x \in C_b} x\| \tag{5.16}$$

- **Ward method:**

$$\frac{|C_a||C_b|}{|C_a| + |C_b|} \|\frac{1}{\|C_a\|} \sum_{x \in C_a} x - \frac{1}{\|C_b\|} \sum_{x \in C_b} x\| \tag{5.17}$$

  which corresponds to a minimization of the cluster variance [153]

Important approaches of this type are the following

- **Agglomerative Hierarchical Nonoverlapping Clustering (SAHN):** In this approach initially each element corresponds to individual clusters. Then, recursively clusters that are pairwise close to each other are merged.
  Complexity: The complexity is $\mathcal{O}(n^3)$ [123]

- **Sequential Divisive Hierarchical Nonoverlapping Clustering (SDHN):** In this approach initially all elements correspond to one cluster. Then, clusters are iteratively split until only one element per cluster remains. Splitting is done using partitioning algorithms such as k-means. For this k-means is executed once per split operation.
  Complexity: The complexity is $\mathcal{O}(2^d nkt)$ [154], where d is the maximal depth of the clustering and t the maximum number of iterations of k-means.
- **Divisive Analysis Clustering (DIANA):** In this approach at each iteration the cluster with biggest diameter is chosen. Within this cluster the element of furthest distance is chosen and defined as a cluster center. Then, all data points of the original cluster are assigned the cluster center that is closest at each split, i.e. the newly assigned center or the original center [155].

Density-based Clusterings: While the above approaches are good in identifying spherical clusters, density based approaches allow to find clusters of complex shapes. For this elements that are close to each other form clouds that represent the cluster. As basis for this a radius ϵ is assigned around each element [151]. Important approaches are the following.

- **Density Based Spatial Clustering of Applications with Noise (DBSCAN):** In this approach initially core objects are defined that consist of *MinPts* elements per cluster. These elements are then, connected if these are reachable within their density region. That is the case if two objects p and q if p is in the ϵ neighborhood of q. The connected regions form the final clustering.
  Complexity: The complexity is $\mathcal{O}(n \log n)$ [151]

Raster-based Clusterings: This type of approaches attempts to optimally place a grid in the space of the elements to be clustered, where each element is part of a cell resulting from that multidimensional grid. Important algorithms include the following.

- **WaveCluster:** This approach uses the wavelet transformation to separate clusters. For this the space is divided in m regions, i.e. m^d cells. Per feature of a data point according to its grid position a wavelet transformation is performed. In the resulting coefficients high frequent regions represent edges of the cluster while low frequent ones are cluster centers. By grouping connected areas in the transformed space clusters are found, which can e.g. be done using graphs as proposed in [157].
  Complexity: The complexity for clustering is $\mathcal{O}(n)$.

Other Clustering Approaches: Further important algorithms that do not fall in above categories are the following.

- **Affinity Propagation:** In this approach messages are exchanged between data points. The value of each message contains the affinity of a data point to choose another data point as its cluster centroid. The algorithm then assesses how well each data point is suited to be chosen as a cluster centroid. In multiple iterations information are exchanged that define how high the likelihood of each data point is to be a centroid. This is done until convergence [126].
  Complexity: The complexity is $\mathcal{O}(n^2)$ [126]

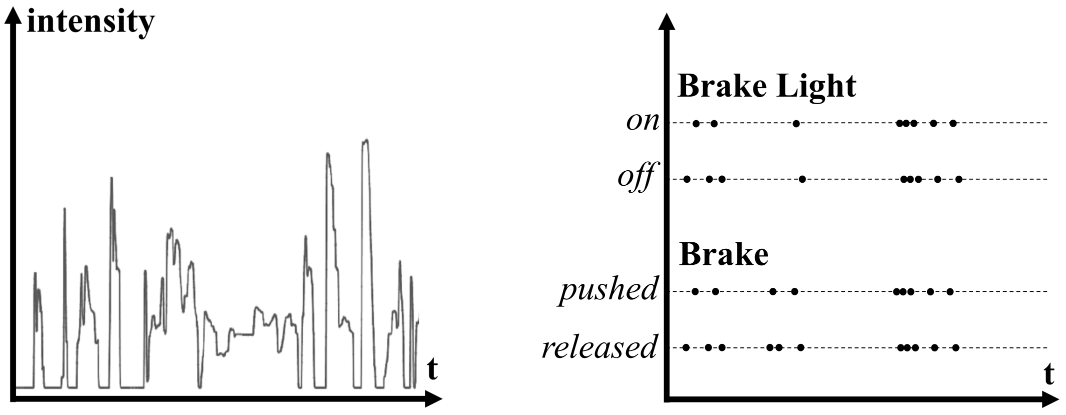
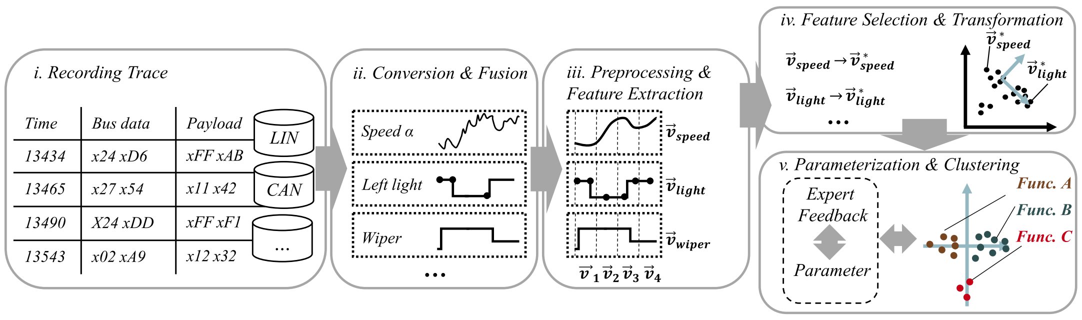


Figure 5.1: Two sequences of TVs are shown. On the left a numerical TV time series is shown and on the right two nominal TVs, with two states each, are shown.

- **Self-Organizing Maps (SOM):** SOMs are q-dimensional regions of l nodes with $q \in \{1, 2\}$. In two dimensions this might be rectangular or hexagonal nodes. Each node i has a positional vector r_i that defines its position on the map and a vector m_i as a reference vector which is a point of the data set. Training is performed by iteratively determining for each element $\mathbf{x} \in X$ the node i whose reference vector is closest to $\mathbf{x}$. Clusters are found by grouping elements using similar reference vectors [127].
  Complexity: The complexity is $\mathcal{O}(ni)$, where i is the number of iterations [158].

5.2.2 Overview

In the following a feature-based approach is presented that is used to cluster TVs of heterogeneous data types and of massive size. The basic idea is to represent each TV using overlapping windows that each contain a set of features. If a TV changes in a common window with a related TV it is assumed that its features are closer to each other at the corresponding points in time. That is, common TVs perform similar actions at similar points in time. For instance, assuming two disjoint TVs, where one variable is always active while the other is not, features at corresponding dimensions would never have values and thus, would be further away and not clustered together.
Resulting from the preprocessing presented in Chapter 4, in the interpreted trace $\mathbf{K}_s$ each TV is either of numerical, nominal, ordinal or binary type. The latter three types are referred to as nominal, as these are treated identically during the clustering procedure. Each numerical TV forms an regularly sampled time-series, while the other types are seen as an event sequence, where only state changes are observed and identical consequent values were filtered out in the preprocessing step. Further, all TVs operate on the same time scale. These representations are used in the following to cluster similar TVs. An example of these signal types is shown in Figure 5.1.


i. Recording Trace
LIN
CAN
...
ii. Conversion & Fusion
Speed α
Left light
Wiper
...
iii. Preprocessing & Feature Extraction
$\vec{v}_{speed}$
$\vec{v}_{light}$
$\vec{v}_{wiper}$
$\vec{v}_1 \vec{v}_2 \vec{v}_3 \vec{v}_4$
iv. Feature Selection & Transformation
$\vec{v}_{speed} \rightarrow \vec{v}^*_{speed}$
$\vec{v}_{light} \rightarrow \vec{v}^*_{light}$
...
$\vec{v}^*_{speed}$
$\vec{v}^*_{light}$
v. Parameterization & Clustering
Expert Feedback
Parameter
Func. A
Func. B
Func. C

Time	Bus data	Payload
13434	x24 xD6	xFF xAB
13465	x27 x54	x11 x42
13490	X24 xDD	xFF xF1
13543	x02 xA9	x12 x32

Figure 5.2: An overview of the overall TV clustering approach is shown [5].

Overview: The basic steps of the clustering approach are shown in Figure 5.2. Observed temporal data is used to extract a feature representation per TV. There overlapping windows are formed and features are extracted per window. In particular features need to be identified that allow to group nominal and numerical TVs. These features are stacked and form the representative of the TV. Next, feature selection and transformation is performed to reduce the dimensionality, before lastly, clustering is performed together with the expert, who aims to determine the appropriate coarseness of clustering as well as the selection of relevant signals that did not end up in identical clusters. This is as clusters are formed based on common occurrence while steady TVs might be of relevance for consequent Specification Mining as well. All steps are discussed in detail in the following Sections. These steps were first presented in [5] and are explained here in more breadth.

5.2.3 Preprocessing

In the preprocessing performed in Chapter 4 the goal was to minimize the loss of information during automated preprocessing, while reducing the trace and preserving the TVs characteristics. For clustering of TVs however, loss of information is allowed, as only the key characteristics need to be considered and comparability of TVs needs to be ensured. For instance, considering TVs such as speed and braking, it is more likely for these TVs to be clustered if these have a value range between 0 and 1, rather than its original units. Therefore, in this step a set of filtering, cleaning and normalization steps are performed to ensure that TVs that change together are clustered together.

Cleaning: Multiple data cleaning operations might be performed at this stage to ensure good data quality for clustering. First, this is the removal of invalid values, which might be resulting from errors in the recordings. For instance if the state was recorded from an automotive networks this might be invalid messages. This is done either through an expert or rule based (e.g. *regular expressions*). Further, missing values might be replaced with its last value if required. For the case of numerical TVs filtering operations are performed that allow to remove outliers. This is only performed if a sufficient number of distinct values for that TV is given. This can be done in multiple ways including using Moving Average, Moving Median or Exponential filtering. It was found that the latter performs best, as other approaches tend to add a time-shift to the data making it unsuitable for grouping according to similar time steps.

Normalization: Further, the aim is to group TVs with similar curves, such as speed and the wheel frequency, which have different ranges but similar shape. To make these comparable each TV curve is normalized to a defined value range. Approaches for normalization include Min-Max normalization, z-transformation or interquartile normalization. It was found that in comparison to the further two approaches, the interquartile normalization is less prone to outliers and thus, is best suited to normalize the given numerical time series.

Symbolic Aggregate approXimation (SAX) Algorithm: Numerical time-series are additionally symbolized using the SAX algorithm, which is shortly described here. In this approach first a Piece wise Aggregate Approximation (PAA) is applied to a time series. That is, a z-transformation is applied, segmentation is performed and per segment a representative value of the time series is found, e.g. the mean of the values in the segment [88]. SAX puts these values in buckets and assigns a nominal value per bucket. The values defining the borders of the bucket are chosen such that it is equivalent to a Gaussian distribution that is divided into chunks of equal size [88].

5.2.4 Feature Extraction

To capture temporal-causal dependence in a feature vector, a windowing approach is applied on the preprocessed trace. This is challenging as no dominant signal is given and thus, the segmentation points are not clear. Further, using PIPs for segmentation might yield windows where numerical TVs change while its corresponding nominal type TVs that change at such points fall in the window before, thus, not clustering such correlated TVs. Thus, to be able to not omit patterns at the borders of windows a static window with overlap is chosen for this task. The size of the window depends on the frequency of changes of the data set and thus, needs to be determined per data set.

Extraction Approach: As shown in Figure 5.3, at first TVs are segmented into such overlapping windows. Next, for each window w_i of each TV s_i a sub-feature vector $v_i = f(w_i, s_i)$ is extracted. This is done using the function $f : (w, s) \mapsto v$ which extracts a set of features based on the data type $type(s_i)$ of the TV. The numerical TVs are referred to as $type(s_i) = num$ and the nominal ones as $type(s_i) = cat$. Notably, nominal signals do only contain state changes at this point and it is assumed that these state changes only occur at areas of functional procedures, i.e. correlating TVs that are part of the same functional procedure tend to change jointly.

Features: The data type defines the characteristics that are relevant for extraction to suitably represent the value behavior of a TV s_i in state and time. Therefore different sets of features are extracted for both types. These are the following.

If the data type is $type(s_i) = num$ the dominant characteristic to capture is the shape of the TV per window, which is the flow of the values and the type of change. To represent this the following features are identified as candidate representatives for the temporal data X of each TV.

- **Mean:** As data was normalized before this value resembles the height of a TV [151], which is

$$\mu(X) = \frac{\sum_{i=1}^{n} x_n}{n} \tag{5.18}$$

- **Variance:** This defines the deviation from the mean and thus, the dispersion of the TV sequence [151], which is

$$\sigma^2(X) = Var(X) = \frac{1}{n}\sum_{i=1}^{n}(x_n - \mu(X))^2 \tag{5.19}$$

- **Skew:** The skew measures the lack of symmetry [109], which is

$$skew(X) = \frac{\mathbb{E}[(X-\mu)^3]}{Var(X)^{3/2}} \tag{5.20}$$

- **Kurtosis:** The kurtosis represents the relative flatness in relation to the normal distribution [109], which is

$$kurt(X) = \frac{\mathbb{E}[(X-\mu)^4]}{Var(X)^2} \tag{5.21}$$

- **Variance of Slope:** This value measures the intensity of noise in the TV sequence, which is

$$\sigma^2(X') = Var(X') = \frac{1}{n}\sum_{i=1}^{n}(x'_n - \mu(X'))^2 \tag{5.22}$$

- **Magnitude of Mean of Slope :** This value indicates the intensity of a TV trend, which is

$$|\mu(X')| = |\frac{\sum_{i=1}^{n} x'_n}{n}| \tag{5.23}$$

- **Maximal Slope :** This value indicates if a peak is found in the given window, which is

$$\max\{|x| | x \in X'\} \tag{5.24}$$

- **Wavelet Coefficients:** k coefficients of a TV sequence resemble low and high frequency representations of the TV. A good choice is to consider only a subset of highest coefficients, as these are used to reconstruct the time-series and thus, have less loss in information.

These numerical features were chosen as they were successfully applied in similar tasks in [109] and [110].

If the data type is $type(s_i) = cat$ the information about the occurrence and value of a TV in each window is given and subsequent identical samples of the same TV are dropped, i.e. TVs that are part of the same functional procedure share similarity of occurrence behavior, which is captured as features. Further, the goal is to also be able to group numerical TVs together with nominal ones. Thus, in order to do so, numerical signals need to be discretized. This is done either manually according to expert input that is familiar with relevant quantization steps or using automated approaches such as Symbolic Aggregate approXimation (SAX) [88], which is the choice here. After symbolization again elements with identical preceding symbols are removed and thus, only relevant state changes remain after discretization. With this, the following set of features is extracted from categorical and discretized numerical TV sequences:

- **Value Change:** This feature defines if a TV changed its value within the given window. If this is the case the value is 1 and 0 otherwise.

- **Change Ratio:** This value is defined as the ratio between the number of changes that occurred within a window and the total number of samples per window. Thus, it measures the relevance of an interval for a certain TV in comparison to the relevance of the interval for all TVs. It is defined as

$$ROC = \frac{\#changes}{\#samplesinw_i} \tag{5.25}$$

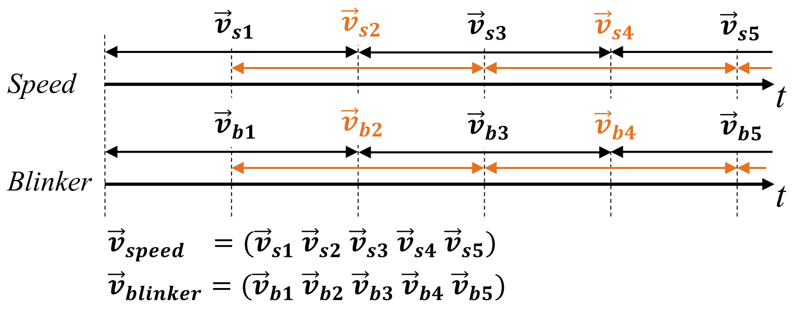
- **Occurrence Frequency:** This value measures the relevance of an interval for a particular TV. It is defined as the ratio of the number of changes of a TV divided by the temporal length of a window. In the given case the window length is constant, which is why this value can be reduced to the enumerator. In general it is defined as

$$OC = \frac{\#changes}{\#time - differenceofw_i} \tag{5.26}$$

- **SAX Mean:** Further, using all SAX values of a TV a distribution of the symbolized values is formed, i.e. A=1, B=2,... . The mean value of this distribution is used as additional features. This mean allows to represent the character of numerical TVs more robustly.

$$OC = \frac{\#changes}{\#time - differenceofw_i} \tag{5.27}$$

Choosing these features allows to compare nominal and numerical signals, while comparison among numerical signals is done on a more fine grained level using its numerical features.



$\vec{v}_{s1}$ $\vec{v}_{s2}$ $\vec{v}_{s3}$ $\vec{v}_{s4}$ $\vec{v}_{s5}$
Speed
t
$\vec{v}_{b1}$ $\vec{v}_{b2}$ $\vec{v}_{b3}$ $\vec{v}_{b4}$ $\vec{v}_{b5}$
Blinker
t
$\vec{v}_{speed} = (\vec{v}_{s1}\,\vec{v}_{s2}\,\vec{v}_{s3}\,\vec{v}_{s4}\,\vec{v}_{s5})$
$\vec{v}_{blinker} = (\vec{v}_{b1}\,\vec{v}_{b2}\,\vec{v}_{b3}\,\vec{v}_{b4}\,\vec{v}_{b5})$

Figure 5.3: The extraction of features from overlapping windows is shown. Per window and TV a set of features is extracted. Per TV these features are stacked to form the feature vector that represents the TV [5].

To now represent a signal s_i with identifier m, sliced in n windows, as a feature vector v_m, all sub-vectors v_{mi} are stacked as

$$v_m = (v_{m1}v_{m2}...v_{mi}...v_{mn}) \tag{5.28}$$

The goal was to find vector representations that both capture the time and value of a TV under heterogeneous data types. In the presented approach this is solved, as the found representation captures temporal interrelation, as same dimensions represent same windows and value behavior is represented by each value in a dimension. Also, comparability between nominal and numerical TVs is enabled by discretization of the numerical time series.

5.2.5 Feature Selection and Transformation

Each resulting vector v_m contains a high-dimensional representation of a TV. To reduce the effect of the curse of dimensionality and to improve computational complexity it is common to reduce dimensionality by selecting relevant features and by transformation of v_m to another space, where less informative dimensions are dropped. This, is especially suitable here, as functional procedures are assumed to occur within certain time spans only, while there are periods of inactivity were no or only less information is transmitted. These periods of inactivity further depend on the TVs that were extracted in Chapter 4, e.g. considering the wheels of a car only information is only given once the car is driving. Reduction of dimensionality is thus, a crucial step. This is discussed in the following.

Feature Selection: Types of selection methods are filter and wrapper methods. Filter methods extract meaningful features independently from the Machine Learning approach that is underlying it. That is, it considers characteristics of the vectors only such as the variance or the eigenvalues. In contrast to that, wrapper based methods do include this information. Based on the performance measured on the learning task, features are chosen, that yield the best metrics [159]. A common approach to determine most important features is to use forward-backward search in a wrapper-based evaluation, i.e. the quality of a subset of features is evaluated on a validation data set using the clustering target (e.g. redundancy or function grouping) it is optimized for. In the

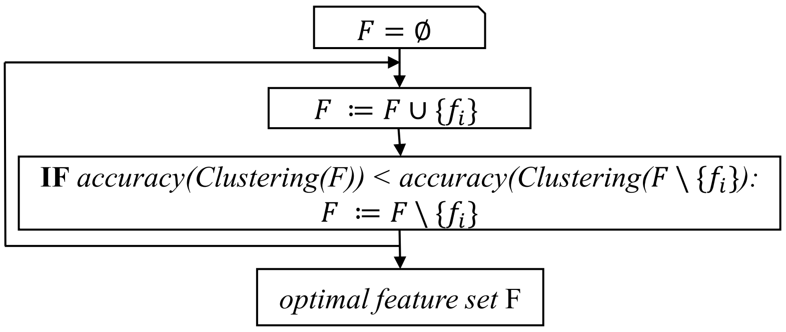

$F = \emptyset$
$F := F \cup \{f_i\}$
IF *accuracy(Clustering(F)) < accuracy(Clustering(F \ {f_i}):*
$F := F \setminus \{f_i\}$
optimal feature set F

Figure 5.4: The forward backward feature selection is shown. At each iteration a feature is added or removed if accuracy improves [5].

proposed Data Mining pipeline this includes either optimizing internal indices, such as the Silhouette index, or using labeled data and external indices, such as a similar data set with expected clustering. Also, by running the approach on data sets with various characteristics a general set of good features is found. As Figure 5.5 shows the selection approach that was chosen consists of two steps called forward and backward selection.

1. *Forward Selection:* Starting from a feature set $F = \emptyset$ and a set of available features F_{all} iteratively features $f_i \in F_{all}$ are added to the set F if their addition yields a better performance than without it [160].

2. *Backward Selection:* Here, a feature is removed from F if this improves the accuracy.

Resulting from this a subset of relevant features is found per data set. The remaining vector however, is still of high dimensionality, as for n windows and f features, the vector has $n \cdot f$ dimensions, which is why a transformation is performed in the following.

Transformation: In Machine Learning it is common to transform data to another representation that is better suited to represent the data. A large variety of approaches for this was introduced in literature. This includes approaches from signal processing, such as discrete Fourier transformations [123] and discrete Wavelet transformations [111] or from Data Mining, such as Piece wise Aggregate Approximation [88], Principal Component Analysis (PCA) or Independent Component Analysis [123].
A two step approach is used to reduce the dimension. In the first, step the variance is computed per dimension. That is if no TV occurred within a window, the variance of all features that correspond to this window will be small or even zero. Thus, low variance dimensions are dropped here. In the second step, PCA is used to transform the vectors to an information maximizing space. As PCA is a linear transformation, inherent properties of each TV vector are conserved. Further, only dimensions with highest information content are used and the remaining dimensions are dropped. With this a significantly reduced vector is found that is used in the further process.

Approach	High-dim.	Complex-shapes	Mult. Gran.	Visualization	Complexity
k-Means	yes	no	yes	no	$\mathcal{O}(nkt)$
k-Medoids	no	no	yes	no	$\mathcal{O}(k(n-k)^2 * t)$
EM	no	no	yes	no	$\mathcal{O}(nk * t)$
DBSCAN	yes	yes	yes	no	$\mathcal{O}(n \log n)$
Agglomerative	yes	yes	yes	dendrogram	$\mathcal{O}(n^3)$
Top-Down	no	yes	yes	dendrogram	$\mathcal{O}(2^d * nkt)$
WaveCluster	indirect	yes	yes	no	$\mathcal{O}(n)$
Affinity Propagation	no	yes	no	no	$\mathcal{O}(n^2)$
SOM	indirect	yes	yes	map	$\mathcal{O}(n * t)$

Table 5.1: Comparison of algorithms in clustering of in-vehicle signals. I.e. handling high-dim. data, detect clusters of any shape, allow multiple granularities of clusters, visual representation and computational complexity, with t iterations, maximal depth d, n examples and k classes. This table was taken from [5].

5.2.6 Formal Comparison of Clustering Approaches

The resulting feature vector might be still of high dimension due to highly complex characteristics per TV. Further, the aim is not only to group TV sequences that occur at exact same times and have exact same shapes. Rather, here the aim is to find TVs that have similar shapes. For instance, lets assume that one dimension is the number of times a signal occurred and another is the peak value of the TV. A spherical clustering would only find peak values that are further apart if also the number of times the TV occurred is further apart. However, it might be the case that the peak value is far away, but still the number of occurrences is similar, which in this case is a good indicator that the TVs correlate on a functional level. This would rather require an elliptical shape of the clustering. Thus, complex shapes have to be supported by the clustering here. Apart from that, complexity of involved TVs is still very high and thus, it is still required to include expert input to find clusters of appropriate granularity. Therefore, it is a desirable property to be able to parameterize the approach towards clusters of certain levels of granularity by adjusting hyper parameters. For instance, at a higher level of granularity per wheel the four sensors of the rotational frequency would be grouped, while at a lower level both left wheel (front and back) and both right wheel sensors could be assigned to two separate groups. Additionally, the computational complexity needs to be kept low and in order to allow for good expert feedback it is important to visualize the data.
Not all of the aforementioned clustering approaches are suited to fill these needs, as is shown in Table 5.1.
Suitability is assessed in these terms as follows.

- *Centroid-based:* Granularity is settable as target clusters k. k-Means is in general suited for high dimensional data as prototypes are found as mean of all clusters and a separation is forced through k. But, only spherical clusters are possible which is contrary to signal feature vectors which can be grouped in any shape. k-Medoids and EM are less suited. In k-Medoids samples are part of the data set which shifts the centroid on a data point and thus, imbalances the center.
- *Hierarchical:* Such approaches are independent of shape, as successive splitting or joining is performed based on neighborhoods. But, top-down clustering tends to split the biggest cluster more often. This results in many clusters of similar size

which is not the target grouping in the given scenario where cluster sizes may vary. Granularity is parameterized on according splitting and joining rules.

- *Density-based:* These approaches allow for multiple granularity by setting the radius per data point, while they are independent of shape as neighboring elements are found using the radius. This radius can exist in any dimension leaving this approach to be well suited for clustering of signals.
- *Grid-based* These approaches allow for multiple granularity by setting the raster size and are independent of shape as the raster can be of any shape. Above that, high dimensional clustering is possible with the limitation that dimensions need to be restricted as e.g. in WaveCluster similar Wavelet coefficients will be too far away to be assigned in one clusters (due to curse of dimensionality). With the reduction to a sufficient number of dimensions and its low computational complexity these approaches are well suited.
- *Affinity Propagation:* Here prototypes are data points themselves, leading to similar imbalance as in k-Medoids. However, common grouping is not dependent on cluster shape as the totality of points is considered for clustering.
- *Self-Organizing Maps:* Due to small numbers of signals each hexagon is sparsely populated by data points making cluster detection difficult.

According to this formal evaluation WaveCluster and DBSCAN are most suited.

5.2.7 Expert Input and Refinement

The process of clustering can be performed in two ways. First, this can be done in a fully automated manner, where hyper parameters are found automatically based on internal indices that assess the quality of clustering. Second, this can be done in an iterative procedure, where the expert is choosing hyper parameters based on a grouping of TVs that is valid and relevant for Specification Mining in his opinion. As clustering itself can be performed effectively once a representation per TV is given and as clusters can be well visualized, this iterative procedure is not excessively time-consuming.
Within this iterative procedure an expert might have three intentions. First, parameters may be adjusted such that a clustering with different goals of grouping are found. Second, given a big cluster the expert might sub-cluster this cluster to get a more fine grained representation. Third, an expert might merge found clusters to get a coarser cluster.
With these clusterings found in a last step the expert is able to select TVs based on the clustering that was found within this procedure. This is required as e.g. steady TVs and active TVs are not clustered together while being still correlated. Also, TVs within multiple clusters might correspond.
Consequently, to optimally leverage the potential of this approach expert input is required, while an automated execution that is based on hyper parameter optimization is possible as well. In the further case, this approach can thus, be seen as a method to structure the TVs and to work as a supporting tool to the expert to identify TVs of common functional procedures that are used for Specification Mining.

Set	Signals (tot[num/nom])	Datapoints (tot[num/nom])	Part of journey
body-id	38 [1/37]	2251 [89/2162]	complete
chassis	53 [18/35]	9999 [9896/103]	start
chassis-nom	35 [0/35]	9896 [0/9896]	start
chassis-num	18 [18/0]	103 [103/0]	start
chassis-ctr	12 [11/1]	10000 [9999/1]	mid
most-freq-num	24 [24/0]	12508 [12508/0]	start
most-freq-ctr	22 [19/3]	11773 [11765/8]	mid
light	39 [6/33]	10055 [2941/7114]	start
mixed	25 [12/13]	69402 [69339/63]	start
mixed-nom	13 [0/13]	9509 [0/9509]	start

Table 5.2: Statistics of the datasets: total number and proportions of numerical and nominal signals, data points per set, recorded part of journey. Here, small subsets are used for evaluation, while in practice thousands of signals are considered. This table was taken from [5].

5.3 Evaluation

So far, a process for clustering of heterogeneous TVs was proposed and multiple methods were compared in terms of suitability for clustering this type of data.
Next, the validity of the proposed approach is evaluated experimentally as it was first presented by us in [5].

5.3.1 Setup

Both preprocessing and feature extraction were performed on a cluster with 70 servers in Apache Spark. The reduced data set was used to perform all further steps including selection, transformation and clustering operations. This was done on a 64-Bit Windows 7 PC with an Intel® Core$^{\text{TM}}$i5-4300U processor and 8 GB of RAM using RapidMiner Studio, Python's Data Mining stack and R.

5.3.2 Data Sets

The statistics of the data sets are shown in Table 5.2. To cover most characteristics of automotive in-vehicle network traces the approach is evaluated on 10 test data sets that are different in terms of TV types (e.g. chassis-nom vs. chassis-num), data points per type, TV number, association to one (e.g. chassis) or multiple (e.g. mixed) functions and resemble different excerpts of a journey. The target of the evaluation is the grouping of TVs in terms of their assignment to similar functions. All approaches were parameterized per data set such that the true number of clusters is achieved and the best possible grouping (according to the expert) within this clustering is reached.

5.3.3 Clustering Criteria

Indices for measuring clustering quality are divided into external and internal indices. External indices use external information, such as a reference clustering. In this case the goal is to ensure cluster homogeneity, i.e. each cluster should consist of elements of a cluster within the reference clustering, and cluster-completeness, i.e. each cluster should contain as many elements of its corresponding reference cluster. Important indices of this group are the Jaccard-Index and the accuracy. Internal indices do not require a reference clustering. According to [161] and [162] the best indices of this type include the Gamma-Index, C-Index, the point based correlation coefficient and the Calinski-Harabasz-Index. For validation the Silhouette-Index was found to be well suited according to [163].
In this evaluation the accuracy and the Silhouette-Index were used.

Accuracy: Accuracy is the number of samples $n_{correct}$ correctly clustered in relation to the total number of samples in the data set $n_{dataset}$ given as

$$acc = \frac{n_{correct}}{n_{dataset}} \tag{5.29}$$

Here, the assignment of reference cluster labels to each TV as a ground truth is done manually by experts.

Silhouette index $s(i)$: This index measures a clustering assignment per data point i in terms of degree of affinity to its assigned cluster relatively to all other clusters. I.e. $a(i)$ as distance of i to all element within its cluster, $b(i)$ as average distance to all data points in all other clusters. It is optimal for $s(i) = 1$ and defined as

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{5.30}$$

5.3.4 Feature Selection

Here, a good feature set for the grouping of TVs in of vehicle systems is sought. For this, the forward-backward feature selection is run on all data sets with its various characteristics (e.g. ratio of numeric to categorical TVs), yielding an optimal feature subset per data set. Next, all features are ranked by counting subsets that contain this feature, which ranks more general features that are valid for more data sets higher. The top ranked features are used for further processing (e.g. top 50). To find features that generalize over all data sets, as a metric the number of times a feature was included in the optimal feature subset, is measured. K-Means was again used for clustering. The results are shown in Figure 5.5. The performance gain of the generalized feature selection was measured before and after the ranking selection, with results shown in Figure 5.6.

Results: It can be seen that for the numerical characteristics best features are the mean, skew, kurtosis, as well as the variance and magnitude of the gradient. This shows that the fine granularity of numerical TV characteristics requires to capture noise, value and shape characteristics. For nominal characteristics all nominal features were suited. This shows that the frequency and type of a nominal/discretized numerical TV is captured.

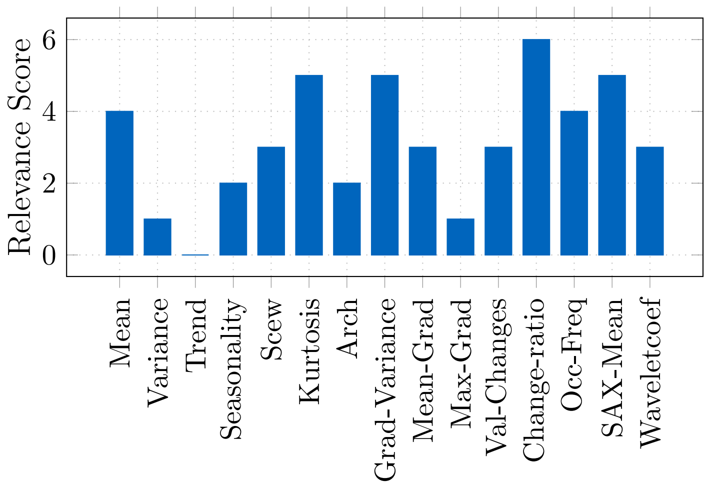


Figure 5.5: Relevance score determined as number of optimal feature subsets in which a feature occurred [5].

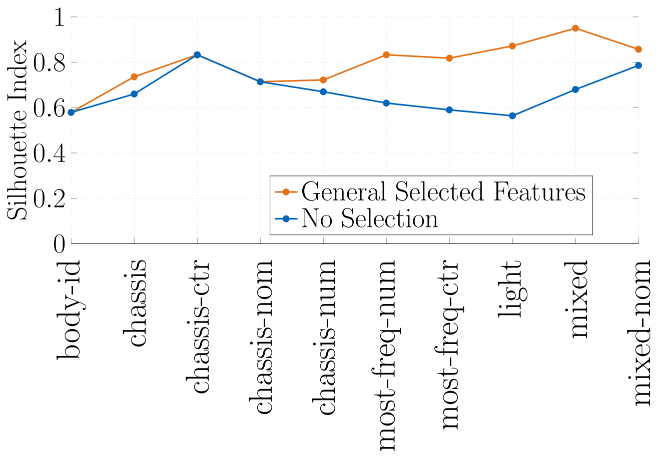


Figure 5.6: Clustering performance in terms of Silhouette index before and after the generalized feature selection is applied [5].

body-id	chassis	chassis-ctr	most-freq-ctr	light	mixed	mixed-nom
128.8	3.5	79.8	1.8	533.6	1.5	2147.7

Table 5.3: Experimentally determined optimal window sizes per data set in seconds. This table was taken from [5].

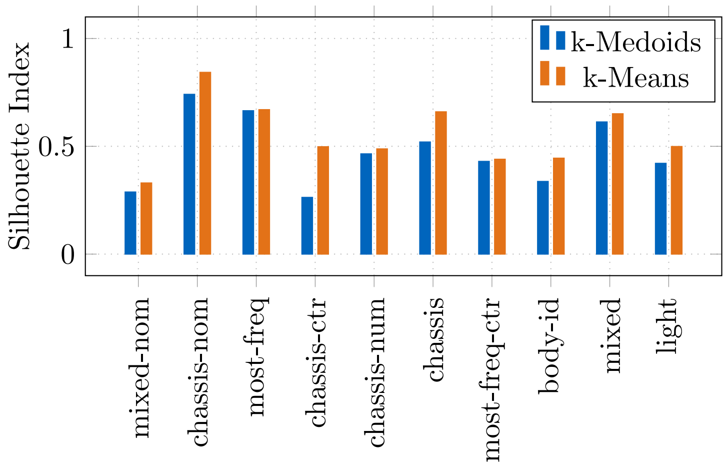


Figure 5.7: Comparison of centroid-based algorithms in terms of Silhouette index [5].

Further, this resembles the assumption that in-vehicle TVs are correlated, when they occur and change their value together. As Figure 5.6 depicts a performance gain of up to 20 % (e.g. at light data set) is achieved with this approach. Notably, all data sets show an improvement after the generalized selection.

5.3.5 Window Size

Setup: After preprocessing each TV was split in windows with 50 % overlap. Based on this all features are extracted, transformation is performed and clustering is applied. Per data set the window size is increased successively from 0.1 seconds to 5000 seconds and the performance is measured in terms of accuracy. From this, the window size with highest accuracy is identified as optimal. Here, K-means was used for clustering, while other approaches yielded similar results. The results are shown in Table 5.3.

Results: If the window is too small patterns relevant for features are overseen, while for big windows feature details are simplified away. Also, as can be seen in table 5.3 less frequently changing TVs, e.g. with a higher number of nominal TVs, require bigger windows , e.g. in body-id, light and mixed-nom, as these TVs do change less often. If more frequently changing numerical TVs need to be clustered smaller windows appear to be optimal which is the case in chassis, chassis-ctr, most-freq-ctr and mixed.

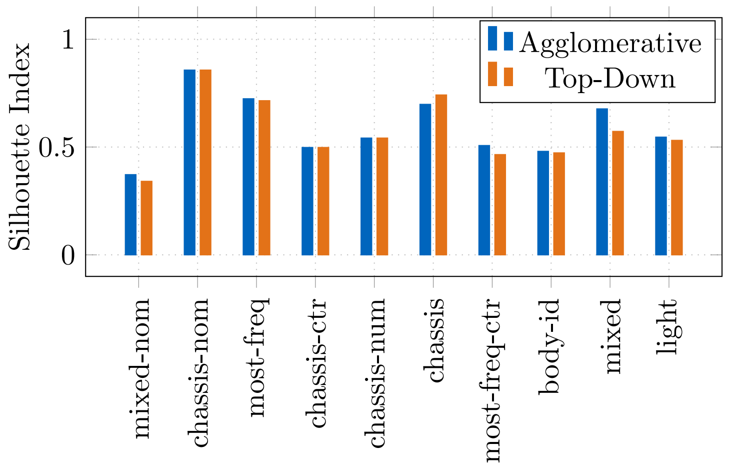


Figure 5.8: Comparison of hierarchical algorithms in terms of Silhouette index [5].

5.3.6 Clustering

The characteristics of in-vehicle TVs require clustering algorithms that can handle high-dimensionality, different granularities and have low computational complexity.

Setup: To examine the suitability of different algorithms for grouping of in-vehicle TVs, k-Means, k-Medoids, DBSCAN, Agglomerative, WaveCluster and SOM clustering approaches were evaluated on all data sets in terms of clustering quality. This is done by using the optimal feature subset as selected by the feature selection approach, parameterization with expert feedback and by consequent application of the clustering approaches.
First, Agglomerative clustering was compared to Top-Down clustering and k-Means to k-Medoids. This is done to evaluated the characteristics of these sub types in terms of applicability to in-vehicle TVs. This is followed by a general experimental comparison of all approaches.

Results - Sub types: As illustrated in Figure 5.7, among centroid-based approaches k-means performs better than k-Medoids. This is, as taking the mean among TVs for clustering avoids a shifting bias.

Results - Hierarchical: Among hierarchical approaches Agglomerative clustering results in better accuracy in 90 % and in better Silhouette index in 70 % of all cases which is shown in Figure 5.8. The best centroid-based and hierarchical approaches are evaluated with further clustering approaches giving results shown in Figure 5.9.

Results - Overall: As depicted in Figure 5.9, DBSCAN, Agglomerative clustering and WaveCluster works best if a data set contains mixed characteristics (i.e. different TV types, proportions of nominal to numerical, etc.) combined. Also, in these cases centeroid-based approaches perform worse. This confirms the expectations and formal analysis of the approaches. Further, as depicted in Figure 5.9, WaveCluster performs

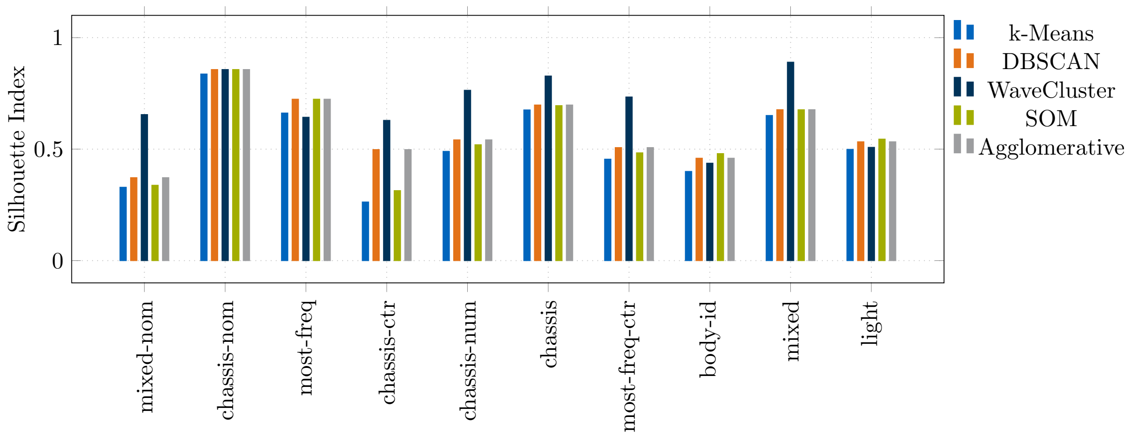


Figure 5.9: Silhouette index per data set and clustering algorithm [5].

best on 80 % of all data sets and shows solid results in the remaining 20 %. Thus, this approach seems best suited for the given scenario. This is because extraction of Wavelet coefficients enables to well capture both fine and coarse grained properties of TVs equally. Also, as described before WaveCluster can well represent the shape and the data's high dimension.
Similarly, DBSCAN and Agglomerative Clustering are well suited to capture these properties. However, the latter approach is biased in that it tends to find clusters of nearly similar sizes which is not given in all test sets.
As deduced before SOM and k-means perform slightly worse, as dimensions are reduced in SOMs and k-means cannot capture varying cluster shapes.

Conclusion: All clustering approaches have solid results in terms of cluster quality. This shows that the proposed processing and clustering approach is well suited for groupings of in-vehicle network TVs. WaveCluster and DBSCAN perform best due to their ability to capture most of the heterogeneous characteristics included in such TVs. As described an optimal window size depends on the structure of the processed data and thus, needs to be determined. Further, feature subsets as discussed in the selection allow for good generalization when clustering in-vehicle TVs.

5.4 Case-Study

In this section it is exemplary shown how the clustering approach can be used to determine TVs of common functions.

Setup: For this case study a realistic data set was used. After the preprocessing of Chapter 4 this data set contains 419 TVs and (after reduction) 20 026 065 data points recorded from one vehicle over eight days. This processing is implemented on a Hadoop system, while the resulting reduced data is processed locally.

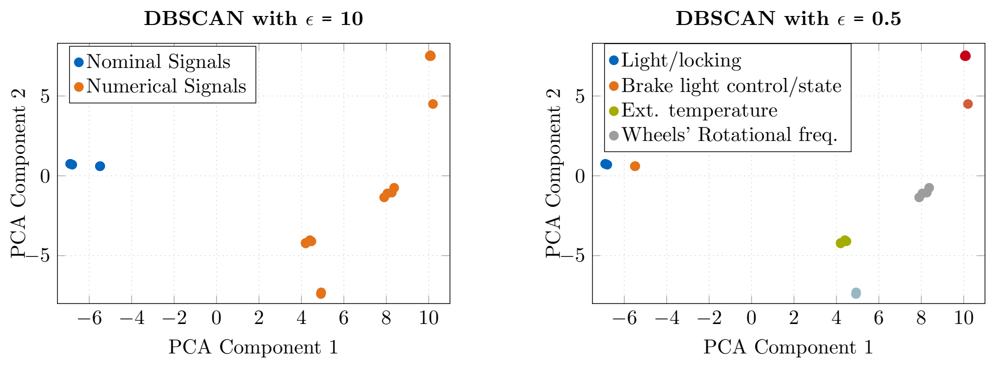


Figure 5.10: On the left the result of signal clustering with DBSCAN and $\epsilon = 10$ and on the right with $\epsilon = 0.5$ are shown [5].

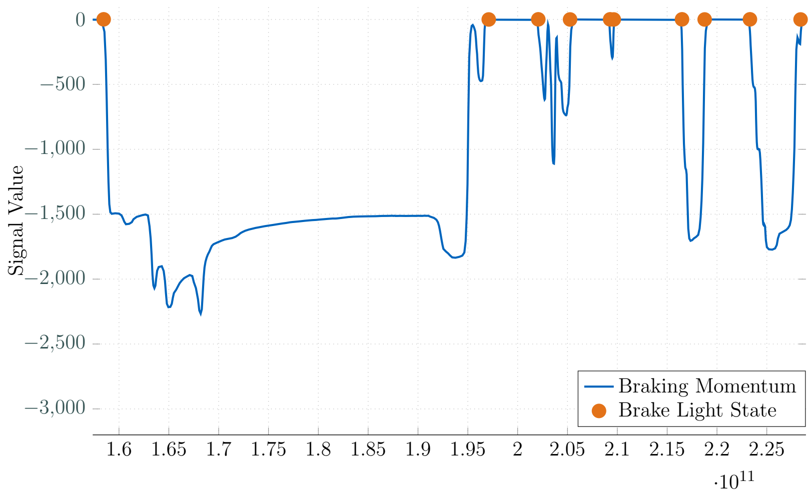


Figure 5.11: The numerical TV "Braking Momentum" and the nominal TV "Brake Light State" are grouped with the presented approach [5].

Preparation: An optimal window size of 17.7 seconds was found with 7 477 windows of 50 % overlap. Per window the features found in the above selection were used resulting in more than 10 000 dimensions per TV. Reduction to less dimensions is done by filtering for dimensions with a variance bigger 0.3 and a successive PCA, resulting in 80 dimensions per TV which are used for local clustering. For clustering DBSCAN was used.

Granularity by cluster inspection: Depending on the parameterization of the clustering, granularity of the target is set. This is illustrated in Figure 5.10 where a coarse grouping separates TVs with different data types and finer clustering extracts TVs of similar functions. Finding an appropriate granularity is done through expert feedback as described above. The extracted clusters are inspected and successively parameterized towards a good target clustering. Experts can then asses the grouping results, e.g. decide whether a grouping signifies a good representation for Specification Mining.

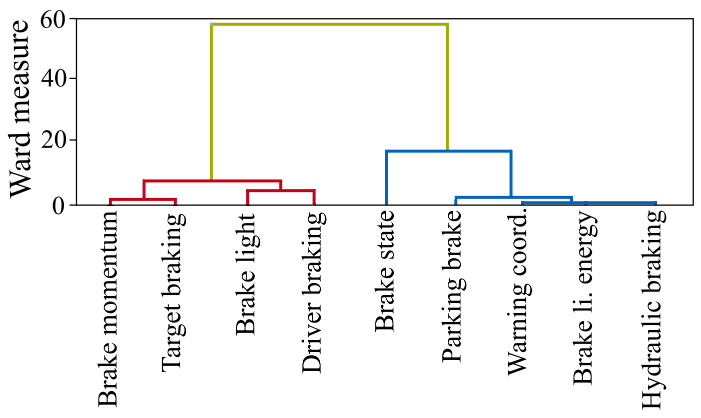


Figure 5.12: Dendrogram illustrating hierarchical clustering at various granularities. I.e. branches resemble possible groupings. E.g. one possible granularity is shown in red and blue [5].

Results: With the presented approach multiple related TVs were found in the analyzed data set. For instance in a fine clustering groupings of speed TVs and TVs representing the time were found. The further were the speed TV for the speedometer, the state of the speed in horizontal direction and the speed of the car's mass center. These are all identical as they measure the vehicle speed, however, for the mining of specifications it could be useful to have these structured in order to decide which once are relevant.
A further example for detected groups of similar functions are TVs related to the braking function which were grouped (see Figure 5.12, red cluster). It shows that the brake light state, state of the driver braking, braking momentum on the wheels and the target braking momentum resulting from the driver pressing the pedal are grouped. In particular as Figure 5.11 shows, with the presented approach nominal TVs were grouped together with related numerical TVs. Further examples of discovered functional groups are TVs for automated parking (e.g. parking space, driver intervention), battery state (e.g. battery capacity, state of charge) or constant TVs (e.g. air pressure, state of the belt buckle). Thus, the proposed approach is well suited to find TVs of common functionality, which in turn enables successive domain-specific Specification Mining of relevant TVs.

5.5 Summary and Conclusion

In this chapter the approach for clustering of TVs is presented. It was shown how TVs of both numerical and nominal type, that are extracted from large scale distributed systems, can be grouped. By using a feature-based approach TV sequences of massive length are broken down to a reduced representation. An optimal clustering is found interactively through expert feedback or automatically through optimization based on internal indices. An evaluation on 10 real world data sets and on a data set with 419 TVs showed that this approach can be effectively applied for this task. Thus, by choosing the right TVs at this stage, the expert is able to perform further semi-automated Specification Mining based on the chosen TVs. By filtering the MSS $\mathbf{K}_n$ and MES $\mathbf{K}_s$ for this subset of TVs the relevant subset trace for Specification Mining $\mathbf{K}_s^k$ and $\mathbf{K}_n^k$ is found. Notably, both $\mathbf{K}_s^k$ and $\mathbf{K}_n^k$ can be based on the selection of a particular cluster, the merging of multiple clusters, as well as the combination of clusters with manually chosen TVs.

6 Concurrent Segmentation and Clustering of Event Sequences

In the previous chapter groups of TVs were identified that correlate in time and state change. Within the trace of this subset of TVs multiple functional procedures might be present, which are unknown a priori. Neither, it is known how many functional procedures are present, nor, it is known to which of those any state change in the overall MSS belongs.
Therefore, in this chapter multiple approaches for identification of functional procedures are discussed, which is referred to as *segmentation clustering* in the scope of this work. This task includes finding clusters of similar behavioral patterns and identifying segments of MSSs that correspond to those patterns in the trace. With this, multiple subsets of similar MSSs are identified from a trace.
Thus, this step uses a trace $\mathbf{K}_n^k$ that was previously reduced in dimension, a segmentation approach D, and hyper parameters of the segmentation approach P_D as inputs. With this, it produces multiple sets of MSSs $\hat{M}$, each representing a set of MSSs M_i of a certain functional procedure, which are formally described as

$$\hat{M} = segcl(\mathbf{K}_n^k, D, P_D) \tag{6.1}$$

The input and output of those approaches are exemplified in Figure 6.1. Notably, in the general case those approaches use MESs as an input. However, those approaches can be used in the same way with MSSs, as the data format is identical when state changes are seen as events.

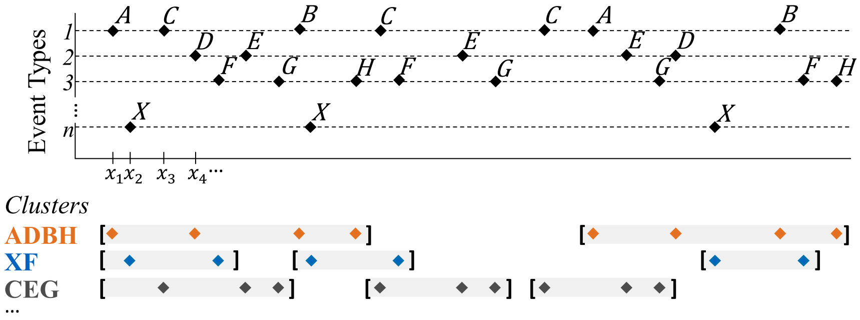


Figure 6.1: Given a MES $< x_1, ...x_N >$ an optimal approach finds patterns that are consistent in state and time. In this case three clusters with representatives *ADBH*, *XF* and *CEG* would be discovered. There, e.g., the cluster *XF* occurs three times within the trace.

Challenges: The task of segmentation clustering is challenging due to the following reasons. First, if overlap between segments is present it needs to be handled, while for the case of no overlap and sparse functional procedures consecutive segmentation and clustering can be performed.
Second, as a result of this clustering segmentation step, resulting clusters might be of bad granularity. This is, as variance within those clusters is dependent on the input data.
Third, in some cases segmentation might inherently require expert input that needs to be included in the designed approaches, e.g. if procedures before a target state are inspected.

Chapter Outline: First, existing approaches of clustering and segmentation and related works are discussed in Section 6.1. In Section 6.2 the problem is formalized. Next, in Section 6.3 extensions of two classical approaches are presented, which are Local Trace Segmentation (LTS) [6] and a window-based approach. Following this, in Section 6.4 a novel large-scale segmentation clustering method is introduced.
All of those approaches yield clusters of MSS sets, which can be further refined by an expert using the method described in Section 6.5. Lastly, in Section 6.6 the discussed methods are compared on a synthetic data set.

6.1 Related Works

Related approaches operate on temporal data, which are either given as time-series or as assumed in this thesis, as sequences. Multiple related fields of research deal with the extraction and grouping of patterns of temporal data. Those are pattern mining approaches, which aim to extract frequent or infrequent patterns, segmentation approaches that cut a trace into segments, clustering approaches for temporal segments, as well as approaches that perform simultaneous segmentation and clustering. In the following an overview of those fields is presented.

Episode Mining: Episode Mining deals with the task of searching for patterns of interest, that are covered within common windows [164]. In general this is similar to segmentation clustering, as it also aims to discover dominant patterns. However, unlike Episode Mining, segmentation clustering aims to assign *all* data points to a segment and a cluster.
The first attempt to this type of mining was sequential mining [165]. There most frequent patterns where mined from sets of sequences (rather than one sequence), if those occurred a sufficient number of times. Later, this was extended to the discovery of all interesting patterns. Those might be frequent patterns, which are extracted with approaches as presented in [166]. Also, this might be mining of closed sequential patterns which was done in [167, 168].
In general such approaches yield a high number of output patterns, which need to be assessed and reduced according to its interestingness. In [169], this is done by representing patterns using partial orders and in [170] by allowing users to specify regular expressions for patterns of interest.
Discovery of episodes in one sequence was first done in [171], where discovered serial

and parallel episodes are patterns that occurred in a window often enough. Other metrics of interestingness that were used in this context include the number of windows that support a pattern [171] or the maximum number of non-overlapping occurrences of an episode [172]. Next, to such window-based approaches, in [173] frequency of items is counted from points in time rather than per window. Apart from the discovery of dominant episodes, in further approaches higher interestingness of episodes is given if deviations from nominal behavior are present, such as in the approaches proposed in [174, 175, 176]. Also, in temporal data interestingness is considered higher if similar durations of patterns are given [177, 178, 179].
Approaches for this type of mining, range from rule-based [180] to probabilistic [181] approaches and aim to detect parallel, serial or composite patterns.
Similar to those techniques, in this chapter multiple approaches are used to identify segments. Those can be interpreted as patterns in the above context, where interestingness is defined as frequent occurrences of patterns, which is referred to as clusters within the scope of this thesis.

Segmentation: This type of method aims to identify segments within a trace that is used for representation of temporal data.
This includes windowing approaches, which were extracted and used in various ways in past research. First, those could be identified in a top-down manner, where long segments are successively broken down into shorter segments [113]. Similarly, this can be done bottom-up where events and segments are successively grouped to larger segments [113]. An example for the latter is Piece wise Linear Representation, where time series of defined length are aggregate such that the variance of an approximation error per unit is minimized [116]. Second, this includes sliding-windows, where fixed size windows are shifted in fixed steps over the temporal data [113]. Third, this might be adjacent or overlapping windows of fixed size. This is useful when grouping segments as overlap allows to avoid errors around segment borders. Such windowing was for example successfully applied in [120], where a feature based approach is used to perform detection of situations in inertial sensor data. Fourth, in [117] it was shown, that fixed length windows might miss out relevant patterns, e.g. if a pattern is spread across multiple windows or patterns vary in length. One possible solution to this is proposed in [118], where time series are cut at points of relevant changes, which are called Perceptually Important Points and are chosen based on sought target patterns. Other solutions use the entropy to solve this [119].
In general window-based approaches are equally suited for numerical, nominal or heterogeneous data types and are thus, are also used in this chapter.
Other approaches cannot be categorized clearly. For instance, this includes the following. In [182] time series are segmented based on its trend, where a genetic optimization algorithm is used to shift boundaries to ensure consistency. In [181] multigrams are used to identify relevant chunks and in [183] characteristic signatures are identified and used for segmentation.

Temporal Clustering: Another field that is relevant for this chapter is clustering of temporal data. This was already discussed in Section 5.1 of Chapter 5. As discussed there, main types of methods include those that operate on raw data, that use features for clustering and methods from the field of representation learning.

Segmentation and Clustering: This step includes approaches that use a combination of segmentation and clustering. Those might be also categorized in either of the above categories (i.e. segmentation, temporal clustering).
This type of methods might be either approaches that run segmentation and clustering consecutively or simultaneously.
The former type of methods includes the following. In [114] fixed length windows are extracted from time series and the mean of features extracted from those. Based on this patterns in time series are indexed. In a similar approach for indexing the sum of differences between successive windows is used for characterization [112]. The work of [115], extracts segments from time series of electricity consumption by using weekly segments. Based on this multiple features such as variance or skew are used to cluster those. 50 % overlapping windows were applied on inertial sensor data in [120] to identify activities, which is done based on static features including mean, energy or entropy of the segments. Other approaches assume underlying probabilistic processes, which are learned from the data. This is done in [184], where time series are modeled as Markov Chains and the parameters of those models that were learned from the data are used as input for hierarchical clustering. Also, in [185] Markov Random Fields where used to capture dependencies, when clustering and segmenting time-series.
Simultaneous segmentation and clustering of temporal event data was also previously done in PM. There, LTS [6] was presented, which defines windows at each data point and successively reduces those until non-conflicting segments are found. In another approach, called global trace segmentation fingerprints are extracted from segments, which are clustered based on frequency [186]. Above that, in [187] an approach to mine longer macro sequence groups from shorter micro sequences using Markov Models is presented.

The proposed DM pipeline uses the latter type of approaches. Thus, in order to provide a good choice for this stage, multiple of those approaches are revised and compared in the following. First, a window-based approach with consequent feature-based clustering is used. Second, in the same way a simple but effective segmentation approach is presented, that uses ranges around data points to identify segments and feature-based clustering to group those. Third, LTS is used and extended for the given scenario.

6.2 Formal Definition and Problem Statement

In this section a formal description of the segmentation clustering problem is given.

6.2.1 Formal Definition

In the previous chapter MSSs were extracted which are now used to identify segments and clusters from those. This is formalized as follows.

Observed sequence: As illustrated in Figure 6.2, an observed sequence of N sequential events is considered, i.e. $\mathbf{X}_{obs} =< x_1, x_2, ...x_N >\in \mathbf{R}^{2\times N}$, where each occurrence $x_i = (s_i, t_i)^T$ consists of a state of an event $s_i \in Cat$ and a time-stamp $t_i \in \mathbf{R}^+$. In the context of the proposed DM pipeline, the observed sequence $\mathbf{X}_{obs}$ refers to the input trace $\mathbf{K}_n^k$, where the state s_i of each observation corresponds to the corresponding TV with its outcome, i.e. $s_i = (E_i.S_k = E_i.s_j)$, and the time t, i.e. $t_i = E_i.t$.

Clusters and Patterns: $\mathbf{X}_{obs}$ is assumed to be generated from K generators, i.e. functional procedures, that each produce I_k instances $\mathbf{X}_k^j, j \in [1, I_k], k \in [1, K]$ of a sequence pattern $\mathbf{X}_k$.

This pattern is assumed as being exact or as approximately similar as in Section 6.4. In the further case, $\mathbf{X}_k \in \mathbf{R}^{2\times N_k}$ is defined by fixed states $< s_{k1}, s_{k2}, ...s_{kN_k} >$ and its time stamps $< t^j_{k1}, t^j_{k2}, ...t^j_{kN_k} >$. For the latter case this fixed state sequence might be varying in terms of noise and variations of functional procedures.

Further, temporal gaps $\Delta t^j_{ki} = t^j_{ki} - t^j_{ki-1}$ are assumed to be each drawn from a set of defined distributions of $\Sigma_{ki}(\Theta_{ki})$ with parameters Θ_{ki}. Such instances of $\mathbf{X}_k$ are called a *pattern*, while the group of all sequences that are produced by the same generator $\mathbf{X}_k$ are considered a *cluster*. For instance, in an automobile a cluster might be the characteristic sequence of events occurring when the engine is started, while each occurrence of this sequence is a pattern.

Cluster sequence: Assuming O_k such patterns of $\mathbf{X}_k$ the cluster's sequence is defined as $\mathcal{X}_k =< \mathbf{X}^1_k, \mathbf{X}^2_k, ...\mathbf{X}^{O_k}_k >$.

Pattern Assignment: The observed sequence $\mathbf{X}_{obs}$ can thus, be seen as a superposition of all K cluster sequences $\mathcal{X}_k$. In order to preserve the information of cluster and pattern assignment, a pattern vector $\mathbf{p_{obs}}$ of patterns $p_i \in \mathcal{P}$ is introduced, that assigns each event j in $\mathbf{X}_{obs}$ a globally unique pattern identifier p_i from the set of all pattern identifiers $\mathcal{P}$ by setting the entry $\mathbf{p}_{obs}[j] = p_i$. The subset of events with the same identifier p_i form the event set $\mathbf{X}^j_k$.

Cluster assignment: The surjective mapping $c_{obs} : p_i \mapsto c_i$ maps each pattern with identifier $p_i \in \mathcal{P}$ to the cluster $c_i \in [1, K]$ it is assigned to. Thus, $\mathbf{p}_{obs}$ and c_{obs} together define the segmentation and clustering of $\mathbf{X}_{obs}$ into K $\mathcal{X}_k$.

This is illustrated in Figure 6.2 where the shapes of the events resemble the clusters and the separation in each line indicates patterns within that cluster. Note, that in order to determine those patterns within one cluster the symbols of the other clusters need to be blanked out. This is especially complex as initially it is not known which symbol contributes to which pattern.

6.2.2 Problem Statement

In the observed sequence $\mathbf{X}_{obs}$ both the pattern assignment $\mathbf{p_{obs}}$ and the cluster mapping c_{obs}, as well as the length of each pattern N_k and the number of clusters K are unknown.
Thus, as exemplified in Figure 6.1 the aim of the approaches introduced in this chapter

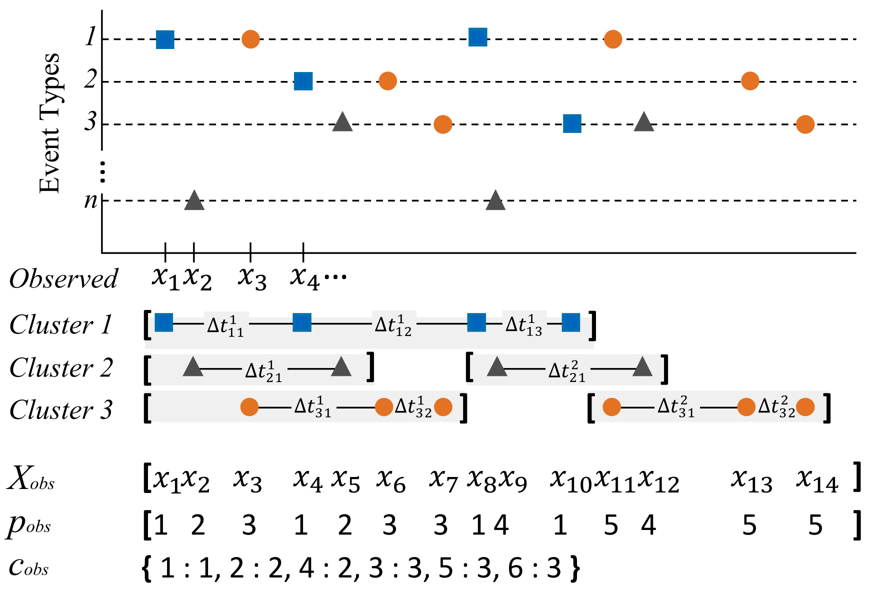

Event Types
1
2
3
n
Observed
$x_1 x_2 \; x_3 \; x_4 \cdots$
Cluster 1
Δt^1_{11}
Δt^1_{12}
Δt^1_{13}
Cluster 2
Δt^1_{21}
Δt^2_{21}
Cluster 3
Δt^1_{31}
Δt^1_{32}
Δt^2_{31}
Δt^2_{32}
X_{obs} $[x_1 x_2 \; x_3 \; x_4 \; x_5 \; x_6 \; x_7 \; x_8 x_9 \; x_{10} x_{11} x_{12} \; x_{13} \; x_{14}]$
p_{obs} [1 2 3 1 2 3 3 1 4 1 5 4 5 5]
c_{obs} {1 : 1, 2 : 2, 4 : 2, 3 : 3, 5 : 3, 6 : 3}

Figure 6.2: An observed sequence $\mathbf{X}_{obs}$ is shown. Here colors indicate cluster assignments. $\mathbf{p}_{obs}$ assigns each event an identifier, which defines the pattern it belongs to. With c_{obs} a cluster is assigned to each pattern.

is to find an optimal pattern and cluster assignment. That is, in order to segment and cluster the observed sequence, the task is to learn those parameters from the observed sequence data. There, patterns might be either overlapping or non-overlapping.

6.3 Extending Existing Approaches

As discussed in Section 6.1 the task of segmentation clustering can be solved partly by existing approaches. In this Section two of such approaches are discussed and extended for the scenario of segmentation clustering. This includes a window-based approach and LTS [6].

6.3.1 Window-based Approach

In this approach the event sequence $\mathbf{X}_{obs}$ is first segmented into non-overlapping windows of size n_w, which are consequently clustered based on its content. To automatically find optimal window sizes the Silhouette index is used.
Initially a good window size is determined by applying the full approach to $\mathbf{X}_{obs}$ with various window sizes $n_w \in R_w$ within a range R_w and by choosing the best n_w based on the Silhouette index. Given those input parameters the following approach is performed.

1. Segment the full trace into windows of size n_w.

2. Per window i compute a vector v_i, where each dimension is the event symbol and the value of each dimension is the number of times this symbol occurs within that window. This yields a set of candidate segments $\mathbf{X}_c$.

3. Next, all vectors v_i are clustered using existing clustering approaches. In this work K-Means, Spectral and Hierarchical clustering were used for this purpose. Based on this grouping of vectors according windows $\mathbf{X}_k^i$ are clustered together, yielding cluster sequences $\mathcal{X}_k$.

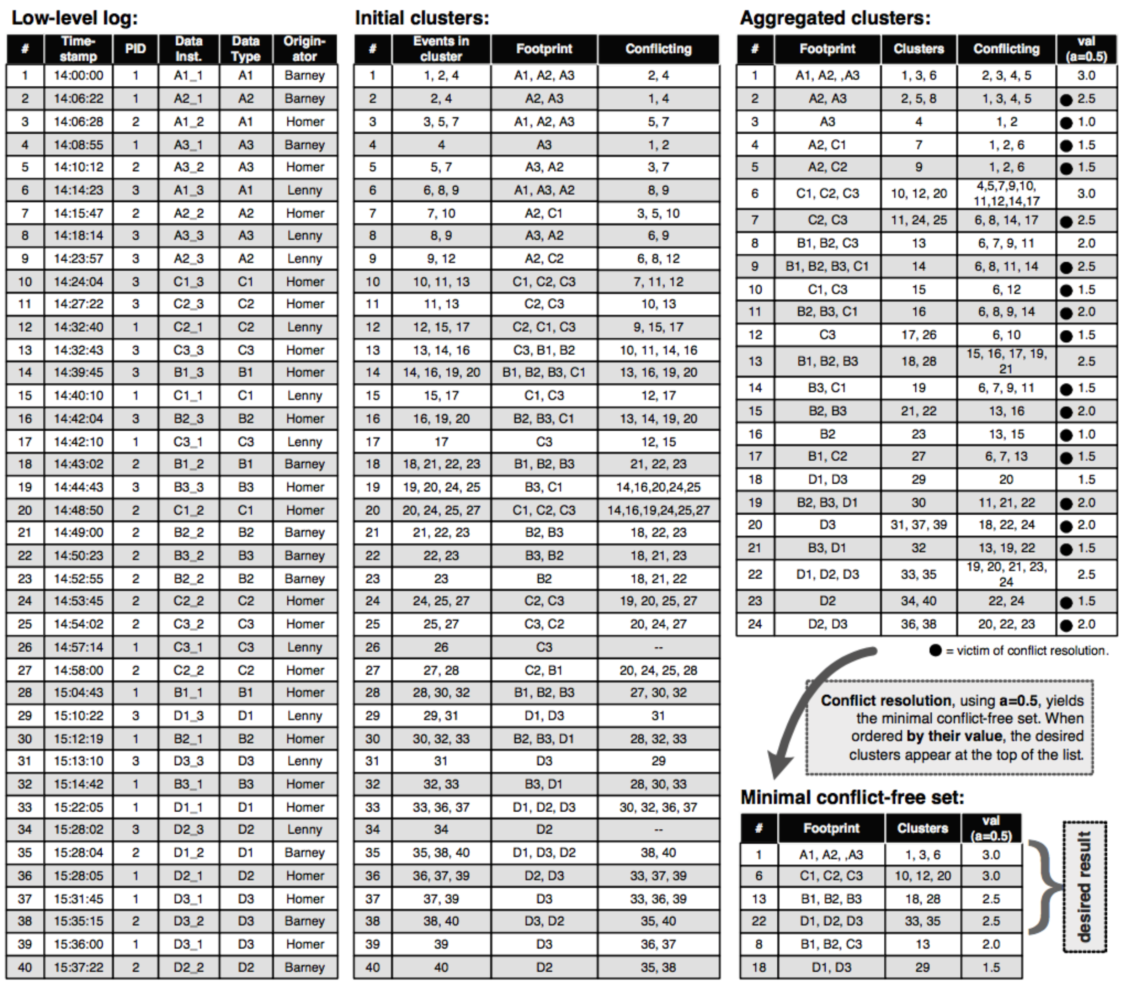
6.3.2 Extending Local Trace Segmentation

Local Trace Segmentation was introduced in PM [6]. As Figure 6.3 exemplifies LTS consists of the following steps.
First, per data point in $\mathbf{X}_{obs}$ all consecutive data points that are within a temporal range of t_{post} are assumed to be candidate patterns (which might overlap). Then, identical candidate patterns are grouped into meta-clusters which aggregate all patterns of the according cluster. Then, from the set of meta-clusters that hold overlapping patterns one cluster has to be chosen for a unique assignment of data points to clusters. For this, a score with parameter α is assigned to each meta-cluster. A larger α prefers meta-cluster with longer patterns. This yields a result set of meta-clusters, which is used to assign all corresponding data points a cluster index and a pattern index.
LTS is not directly applicable to the given scenario due to the following reasons. While in its original version it allows to find most active clusters, it will potentially exclude further clusters. With this not all data points in $\mathbf{X}_{obs}$ are assigned a pattern. Therefore, LTS is extended by running it multiple times on subsets of data points that were not yet assigned.
Furthermore, LTS yields multiple active clusters that might be of similar shape. Therefore, to provide more distinctive cluster representatives another clustering run is used on the set of active patterns.
The extensions yield the following LTS approach that is used here.

1. First, the LTS approach is applied on $\mathbf{X}_{obs}$. This yields a set of active clusters and a subset of pattern assignments $\mathbf{p}'_{obs}$.

2. Next, these found clusters group segments only if those are exactly similar. But those clusters might differ in less symbols only and are thus, very fine grained. Therefore, in this work each active cluster is represented as vector in the same way as described in Section 6.3.1 and consequently clustered using K-means clustering. This yields a cluster assignments c'_{obs} for each pattern in $\mathbf{p}'_{obs}$.

3. In the next stage patterns are dropped and thus, not all data points in $\mathbf{X}_{obs}$ are assigned a pattern and cluster. With this, multiple segments within the remaining data sets might contain sub sequences of the found patterns and clusters c'_{obs}. Therefore, the LTS approach is reapplied on all data points in $\mathbf{X}_{obs}$ that were not assigned. This yields further active clusters. Using the trained K-means model those active clusters are assigned to the closets active cluster that was found in the first step yielding further entries for $\mathbf{p}'_{obs}$ and c'_{obs}.



Low-level log:

#	Time-stamp	PID	Data Inst.	Data Type	Origin-ator
1	14:00:00	1	A1_1	A1	Barney
2	14:06:22	1	A2_1	A2	Barney
3	14:06:28	2	A1_2	A1	Homer
4	14:08:55	1	A3_1	A3	Barney
5	14:10:12	2	A3_2	A3	Homer
6	14:14:23	3	A1_3	A1	Lenny
7	14:15:47	2	A2_2	A2	Homer
8	14:18:14	3	A3_3	A3	Lenny
9	14:23:57	3	A2_3	A2	Lenny
10	14:24:04	3	C1_3	C1	Homer
11	14:27:22	3	C2_3	C2	Homer
12	14:32:40	1	C2_1	C2	Lenny
13	14:32:43	3	C3_3	C3	Homer
14	14:39:45	3	B1_3	B1	Homer
15	14:40:10	1	C1_1	C1	Lenny
16	14:42:04	3	B2_3	B2	Homer
17	14:42:10	1	C3_1	C3	Lenny
18	14:43:02	2	B1_2	B1	Barney
19	14:44:43	3	B3_3	B3	Homer
20	14:48:50	2	C1_2	C1	Homer
21	14:49:00	2	B2_2	B2	Barney
22	14:50:23	2	B3_2	B3	Barney
23	14:52:55	2	B2_2	B2	Barney
24	14:53:45	2	C2_2	C2	Homer
25	14:54:02	2	C3_2	C3	Homer
26	14:57:14	1	C3_1	C3	Lenny
27	14:58:00	2	C2_2	C2	Homer
28	15:04:43	1	B1_1	B1	Homer
29	15:10:22	3	D1_3	D1	Lenny
30	15:12:19	1	B2_1	B2	Homer
31	15:13:10	3	D3_3	D3	Lenny
32	15:14:42	1	B3_1	B3	Homer
33	15:22:05	1	D1_1	D1	Homer
34	15:28:02	3	D2_3	D2	Lenny
35	15:28:04	2	D1_2	D1	Barney
36	15:28:05	1	D2_1	D2	Homer
37	15:31:45	1	D3_1	D3	Homer
38	15:35:15	2	D3_2	D3	Barney
39	15:36:00	1	D3_1	D3	Homer
40	15:37:22	2	D2_2	D2	Barney

Initial clusters:

#	Events in cluster	Footprint	Conflicting
1	1, 2, 4	A1, A2, A3	2, 4
2	2, 4	A2, A3	1, 4
3	3, 5, 7	A1, A2, A3	5, 7
4	4	A3	1, 2
5	5, 7	A3, A2	3, 7
6	6, 8, 9	A1, A3, A2	8, 9
7	7, 10	A2, C1	3, 5, 10
8	8, 9	A3, A2	6, 9
9	9, 12	A2, C2	6, 8, 12
10	10, 11, 13	C1, C2, C3	7, 11, 12
11	11, 13	C2, C3	10, 13
12	12, 15, 17	C2, C1, C3	9, 15, 17
13	13, 14, 16	C3, B1, B2	10, 11, 14, 16
14	14, 16, 19, 20	B1, B2, B3, C1	13, 16, 19, 20
15	15, 17	C1, C3	12, 17
16	16, 19, 20	B2, B3, C1	13, 14, 19, 20
17	17	C3	12, 15
18	18, 21, 22, 23	B1, B2, B3	21, 22, 23
19	19, 20, 24, 25	B3, C1	14,16,20,24,25
20	20, 24, 25, 27	C1, C2, C3	14,16,19,24,25,27
21	21, 22, 23	B2, B3	18, 22, 23
22	22, 23	B3, B2	18, 21, 23
23	23	B2	18, 21, 22
24	24, 25, 27	C2, C3	19, 20, 25, 27
25	25, 27	C3, C2	20, 24, 27
26	26	C3	--
27	27, 28	C2, B1	20, 24, 25, 28
28	28, 30, 32	B1, B2, B3	27, 30, 32
29	29, 31	D1, D3	31
30	30, 32, 33	B2, B3, D1	28, 32, 33
31	31	D3	29
32	32, 33	B3, D1	28, 30, 33
33	33, 36, 37	D1, D2, D3	30, 32, 36, 37
34	34	D2	--
35	35, 38, 40	D1, D3, D2	38, 40
36	36, 37, 39	D2, D3	33, 37, 39
37	37, 39	D3	33, 36, 39
38	38, 40	D3, D2	35, 40
39	39	D3	36, 37
40	40	D2	35, 38

Aggregated clusters:

#	Footprint	Clusters	Conflicting	val (a=0.5)
1	A1, A2, ,A3	1, 3, 6	2, 3, 4, 5	3.0
2	A2, A3	2, 5, 8	1, 3, 4, 5	● 2.5
3	A3	4	1, 2	● 1.0
4	A2, C1	7	1, 2, 6	● 1.5
5	A2, C2	9	1, 2, 6	● 1.5
6	C1, C2, C3	10, 12, 20	4,5,7,9,10, 11,12,14,17	3.0
7	C2, C3	11, 24, 25	6, 8, 14, 17	● 2.5
8	B1, B2, C3	13	6, 7, 9, 11	2.0
9	B1, B2, B3, C1	14	6, 8, 11, 14	● 2.5
10	C1, C3	15	6, 12	● 1.5
11	B2, B3, C1	16	6, 8, 9, 14	● 2.0
12	C3	17, 26	6, 10	● 1.5
13	B1, B2, B3	18, 28	15, 16, 17, 19, 21	2.5
14	B3, C1	19	6, 7, 9, 11	● 1.5
15	B2, B3	21, 22	13, 16	● 2.0
16	B2	23	13, 15	● 1.0
17	B1, C2	27	6, 7, 13	● 1.5
18	D1, D3	29	20	1.5
19	B2, B3, D1	30	11, 21, 22	● 2.0
20	D3	31, 37, 39	18, 22, 24	● 2.0
21	B3, D1	32	13, 19, 22	● 1.5
22	D1, D2, D3	33, 35	19, 20, 21, 23, 24	2.5
23	D2	34, 40	22, 24	● 1.5
24	D2, D3	36, 38	20, 22, 23	● 2.0

● = victim of conflict resolution.

Conflict resolution, using **a=0.5**, yields the minimal conflict-free set. When ordered **by their value**, the desired clusters appear at the top of the list.

Minimal conflict-free set:

#	Footprint	Clusters	val (a=0.5)
1	A1, A2, ,A3	1, 3, 6	3.0
6	C1, C2, C3	10, 12, 20	3.0
13	B1, B2, B3	18, 28	2.5
22	D1, D2, D3	33, 35	2.5
8	B1, B2, C3	13	2.0
18	D1, D3	29	1.5

Figure 6.3: The LTS approach is shown given an example trace as it was introduced in [6].

4. After the previous step still unassigned elements might result. Thus, step 3 is repeated until all data points in $\mathbf{X}_{obs}$ are assigned.

This approach cannot perfectly handle overlapping patterns. However, when an appropriate value for t_{post} is given overlapping patterns are found. This results, as such patterns will treat the overlapping part of another pattern as noise. However, the extended LTS cannot find patterns of various lengths. That is, the length of the pattern is limited by t_{post}.

6.4 Large-Scale Reduced Segment Clustering

Another approach for finding patterns from traces of high length is presented in this section, where non-overlapping segments are assumed. This approach performs segmentation and clustering separately as follows.

6.4.1 Overview

In the case of non-overlapping patterns, patterns with the same identifier p_i are assumed to be neighboring.
Therefore, here the task of segmentation clustering is split into identification of segments p_i followed by clustering of segments based on its characteristics. This is done in the following manner.
During *Range Segmentation*, each observed occurrence x_i is assigned a temporal area of range r_{temp} around it. Then, all occurrences that have an overlapping temporal area are assigned the same segment index p_i. After that, during *Frequency Clustering* all segments with same index are represented as a vector. Using clustering approaches vectors of similar shape are grouped which results in a mapping c_{obs} for each segment index p_i.
Those steps are detailed in the following.

6.4.2 Range Segmentation

In the first step potential segments need to be identified. As stated in Section 6.1 this is done in multiple ways, e.g. by using windowing approaches or forward segmentation as in LTS. For massive traces this needs to be performed in a scalable manner.
In the given scenario MSSs are assumed where functional procedures are sparsely distributed. Events belonging to the same functional procedure occur within a close proximity with a gap of bigger length between segments of distinct functional procedures. The following segmentation is applied.

1. Each event $x_i = (s_i, t_i)$ of the observed sequence $\mathbf{X}_{obs}$, is assigned a region $r_i = (s = t_i - r_{temp}, e = t_i + r_{temp})$.
2. Next, all neighboring events r_i and r_j whose regions are overlapping, i.e. $r_j.s \leq r_i.e$ are assigned a unique segment index p_i. Notably, with this multiple events that are within a close proximity share the same segment index.
3. As each event represents a state change of a TV concatenation of all events with the same segment index results in a set $\mathcal{M}_{cand}$ of MSSs $M_i = \langle E_1, ...E_l \rangle$.
4. As the aim is the identification of dominant behavior and specifications, a minimum length l_{min} of a MSS sequence needs to be given in order for the sequence to contain meaningful procedural information. Thus, within $\mathcal{M}_{cand}$ all M_i of length $|M_i| \leq l_{min}$ are dropped giving a reduced set $\mathcal{M}'_{cand}$.

With this, a set of segments $\mathcal{M}'_{cand}$ results.

Discussion: Notably, as the previous step of the DM pipeline reduced the trace to relevant dimensions, less spurious events are contained that could blur this segmentation. Noisy events do rarely create bridges between two segments. Further, by defining the range r_{temp} the coarseness of functional groupings can be adjusted based on expert input. Two parameters r_i and l_{min} are required in this approach.
Choosing a small range r_i results in many fragmentary events, as no neighboring events

are found for concatenation. As opposed to this, a too high value results in segments of big size that potentially contains mixtures of multiple functional procedures and thus, lacks quality of segmentation clustering. Consequently, a good value for r_i lies in between of those two extremes and needs to be chosen such that it provides a segmentation of multiple comparable clusters.
l_{min} is meant to reduce noise during extraction, as a higher value for this parameter reduces the candidate set. Also, often an appropriate r_i still produces a number of fragmentary segments. Thus, l_{min} helps to extract dominant behavior, while keeping an r_i that yields fine granularity of segments.

6.4.3 Frequency Clustering

The set $\mathcal{M}'_{cand}$ contains segments of multiple functional procedures at this point. Thus, in this step clusters need to be identified. For this the same clustering approaches as described in Chapter 5 are used, where DBSCAN is chosen here, as it is well able to capture complex structures and does not require to known the target number of clusters a priori. This is done as follows.

1. Each segment M_i of $\mathcal{M}'_{cand}$ is part of a functional procedure c_i. To be able to perform feature-based clustering on those segments, feature extraction is performed per M_i. For this, it is assumed that segments with TVs that change its state similarly often correspond to same functional procedures. With this, neither the state value of each TV is included nor is the order of the events in M_i. The further is required as the aim is to find variants of functional procedures that correspond to multiple patterns of TVs, e.g. in a car pressing the TV brake might slow down the TV speed while not pressing it might not slow it down. The latter can be excluded due to similar reasons. However, it is assumed that including order would improve clustering precision. Nevertheless, using order would introduce additional computational and structural complexity that needs to be dealt with. This is especially badly suited for the case of massive numbers of segments which is why the focus is on capturing the behavior based on frequencies only.

2. The following feature extraction is performed per M_i. Assuming that in $\mathcal{M}'_{cand}$ a set $\mathcal{S} = \{S_1, ...S_n\}$ of TVs S_k is present, a vector v_i of dimension n is used per segment, where each dimension corresponds to the number of times a TV is observed in segment M_i. This results in a set of feature vectors $\Sigma = \{v_1, ...v_n\}$ For instance assuming a sequence $\langle A = a, B = b, A = c, C = d, C = a\rangle$ yields a vector $(2, 1, 2)^T$ if the order of dimensions defined by its TVs is A, B, C.

3. For high dimensional spaces PCA might be used to optimize the clustering performance. Further, by filtering according to variance per dimension TVs that remain steady and thus, do not contribute to the characteristics of each segment can be removed. This step is optional. Notably, if no PCA is applied prior to this step the pairwise distance between vectors simply represents the absolute number of TVs that differ in those segments.

4. Using DBSCAN and its parameters ϵ_{SC} and $MinPts_{\text{SC}}$ with this clusters are identified. Further, the Manhattan distance is used. If no PCA is used with this ϵ_{SC} specifies the number of TVs that are maximally allowed to differ between segments such that those are assumed part of the same functional procedure. Applying clustering results in k clusters c_i and assignments c_{obs} of those c_i to segments with p_i that identify sequences in the trace in p_{obs}.

5. By grouping all MSSs of identical c_i the output of this stage are multiple sets of MSSs M_i that each can be seen as observed instances of functional procedures.

With this, sets of MSSs that can be used for Specification extraction are identified.

Discussion: As stated before, neither order, nor states of TVs are included at this step. Instead, frequency of TVs and clustering are used to represent each segment. While this might decrease clustering performance, it allows to perform segmentation and clustering at a large scale. This is, as the extraction of features, as well as the reduction of dimensions works well in parallel.
Granularity of this clustering is defined by the parameterization of ϵ_{SC} and $MinPts_{\text{SC}}$. Bigger values for ϵ_{SC} result in functional procedures of higher diversity, while lower values yield nearly identical procedures only. $MinPts_{\text{SC}}$ is a filtering criterion that ensures only most dominant clusters to be found.
In contrast to LTS, this approach cannot handle overlapping patterns as close data points are grouped together. However, it is well able to find patterns of varying lengths. In LTS the length of the learnable segment is determined and thus, limited by t_{post}, while in range segmentation any length is possible as long as data points are within a range r_{temp}.

6.5 Refinement Clustering

All of the presented approaches yield clusters of segment, which might potentially be too coarse or too fine grained, as characteristics of input traces and pattern clusters might vary. Also, similar to the case of clustering TVs, clustering for functional procedures has no uniquely correct separation. Rather, an appropriate clustering granularity is within the discretion of the expert, as, e.g., a useful granularity of clusters might be different per cluster that was found.
Therefore, to refine the outcome clusters of the previous approaches, as an optional step, expert-based refinement of segment clusters is performed. This postprocessing consists of the following steps.

1. **Aggregation:** Given a set of K clusters of segments, experts can combine multiple clusters yielding an aggregated cluster.

2. **Segmentation:** By performing successive subclustering, the clusters c_i are further subgrouped into more fine grained clusters.

Both steps can be performed visually, e.g., by inspection of cluster distributions. As an alternative to expert input, refinement can also be done in an automated manner. For this appropriate metrics need to be defined and automated sub clustering in a top down manner needs to be performed until an implicit quality metric is improved.

6.6 Evaluation

In this section an evaluation of the above approaches is presented. This is done synthetically by evaluating the performance in terms of run time and assignment quality.

6.6.1 Setup

Implementation: All approaches are implemented in Python and the experiments are conducted on an HP™ Z-840 equipped with two Intel® Xeon® E5-2640 v3 2.60GHz CPUs and 96 GB of RAM.

Data Generation: A data generator is used to synthetically generate event sequences of different patterns, which are used as input to all approaches. The generated event sequence $\mathbf{X}_{obs}$ has the format described in Subsection 6.2.1 and is created as follows.

1. A defined set of distinct symbols Ξ_C of size $|\Xi_C|$ is generated.
2. Generate K distinct template sequence patterns $\mathbf{X}_k$, that each represent the characteristic pattern of a cluster. Those are determined as follows.
   a) Given a target symbol size $|\Xi_k|$ find a set of candidate symbols $\Xi_k \subseteq \Xi_C$ of size $|\Xi_k|$ for $\mathbf{X}_k$. For this, first, determine r_C percent of characters in Ξ_C that will be common in all candidate sets Ξ_k and by second, drawing the remaining candidates from Ξ_C to get candidate symbols of size $|\Xi_k|$. The length $|\Xi_k|$ is drawn from $|round(\mathcal{N}(\mu_{len}, \sigma_{len}^2))|$.
   b) Draw a target sequence length $|\mathbf{X}_k|$ per pattern from $\mathcal{N}(\mu_{seq}, \sigma_{seq}^2)$, and sequentially choose symbols from Ξ_k at random (with repetition of symbols) to form the pattern $\mathbf{X}_k$. To find various temporal gaps Δt_{ki}^j, a mean distance per cluster $\mu_{k\Delta t}$ is drawn from $\mathcal{N}(\mu_{\Delta t}, \sigma_{\Delta t}^2)$ and a fixed $\sigma_{\Delta t}^2$ is chosen. Then, for each gap Δt_{ki}^j a gap is chosen from $\mathcal{N}(\mu_{\Delta t}, \sigma_{\Delta t}^2)$.
3. Given this set X_T of template patterns $\mathbf{X}_k$ the observed sequence $\mathbf{X}_{obs}$ of length $|\mathbf{X}_{obs}|$ is generated by concatenating template patterns that are randomly drawn from its candidate set, where consecutive template patterns are not identical. The length $|\mathbf{X}_{obs}|$ is determined by drawing $K \cdot n_{dat}$ patterns from X_T.
   To simulate overlap between patterns an overlap ratio r_O is defined. Each consecutive pattern then, starts at the time of the q^{th} symbol of the previously drawn pattern shifted by a random time. This random rime is chosen to be 20 % of the average gap size between elements in $|\mathbf{X}_k|$. q is defined as $q = |\mathbf{X}_k| - (r_O \cdot |\mathbf{X}_k|)$. Further, to simulate completely non-overlapping segments a negative overlap ratio r_O defines a temporal shift of $|r_O|$ times the average gap size as the distance between two $\mathbf{X}_k$ in $\mathbf{X}_{obs}$.
4. The resulting sequence $\mathbf{X}_{obs}$ is used for evaluation.

The ground truth of pattern assignments $\mathbf{p}_{obs}$ and cluster assignments c_{obs} for $\mathbf{X}_{obs}$ is known with the above procedure and thus, is used for evaluation.

Metrics: To validate the quality of the approaches the true assignments of patterns $\mathbf{p}_{obs}$ and clusters c_{obs} are compared to its estimates $\mathbf{p}^*_{obs}$ and c^*_{obs}.
In order to compare these assignments in a fair manner, pattern and cluster assignment indexes are renamed to hold the assignment with biggest overlap with the ground truth, e.g. a pattern $\mathbf{p}^*_{obs} = (8884442)$ and $\mathbf{p}_{obs}. = (1111222)$ is renamed to (1112223), as the overlap between 8 and 1 is 75 % and between 4 and 2 is 66 %. Patterns that are present in $\mathbf{p}_{obs}$ or $\mathbf{p}^*_{obs}$ but not in the respective counterpart (e.g. 3 here) are added as zero occurrence to it in order to allow for computation of classification metrics. With this Precision P_p, Recall R_p and the F1 Score F_p are computed between $\mathbf{p}_{obs}$ and $\mathbf{p}^*_{obs}$. The same values are computed for the clustering assignments c_{obs} and c^*_{obs} as P_c, R_c and F_c. Lastly, computational complexity is measured using the run time.

Compared Approaches: The presented window-based approaches, LTS [6] and the Large Scale Reduced Segment Clustering are compared here, where the following parameterization is used.

- **Window-based Segmentation Clustering:** Three variants of this approach are used here. Those vary only in the type of clustering that was chosen. These are K-Means, Spectral and Hierarchical clustering. K is chosen to be the ground truth number of clusters. The optimal window size is found by exhaustive search within a range of the ground truth pattern length, where the size that maximizes the Silhouette index [191] is used.

- **Variant of LTS:** Parameters of this approach include α, where a larger α prefers meta-cluster with longer patterns. Further, K defines the target clusters for clustering. For the given experiments $\alpha = 0.7$ is chosen and K is set to the ground truth. Further, t_{post} is chosen as $\mu_{len} \cdot (\mu_{\Delta t} + \sigma^2_{\Delta t})$, i.e. the expected duration of the sequence.

- **Large Scale Reduced Segment Clustering:** To allow for clustering based on TVs all data points are assigned a TV with the identical name as its symbol. Thus, rather than TVs, here symbols are clustered. For DBSCAN $\epsilon = 2.5$ is used and no PCA is applied. Lastly, the temporal range is chosen such that the gap between two elements is covered using the range r_{temp} of the algorithm. Thus, $r_{temp} = 0.55 \cdot (\mu_{\Delta t} + \sigma^2_{\Delta t})$ was chosen.

6.6.2 Performance Comparison

In this subsection the results of performance evaluation in terms of classification performance and run time of all approaches is presented. For this individual parameters of the data generator where varied to show individual aspects of the performance. An optimal approach is both able to handle overlap and to scale well.

Overlap Robustness: First, the overlap robustness of the given approaches is compared by running these with different ranges of overlap of generated sequences. Starting from small values which corresponds to a clear separation of sequences ($r_O \in [-10, -2]$), ranges are increased until those eventually yield overlapping sequences ($r_O \in]0, 0.6]$).

The resulting F1 scores computed for clustering and pattern assignment are given in Figure 6.4.
Parameters: $K = 5$, $n_{dat} = 75$, $r_C = 0.2$, ($\mu_{len} = 10$, $\sigma^2_{len} = 2$), ($\mu_{seq} = 10$, $\sigma^2_{seq} = 0.5$) and ($\mu_{\Delta t} = 6, \sigma^2_{\Delta t} = 0.5$).
Results: For small values of r_O the data has a clear gap. As the F1 score shows high values in this region, it can be seen that for non-overlapping inputs, assignments of high accuracy are estimated by all approaches. LTS performs best, which is due to the fact that clear gaps allow LTS to identify the exact segments. Range segmentation is slightly worse as closely neighboring elements might have been assigned the wrong pattern given the chosen parameterization. For the window-based approaches most sequence elements are grouped in the right window. Nevertheless, it might occur that the window is cut at a point in time that falls within a pattern. As a result of this, elements of one pattern end up in different windows, which decreases both pattern and clustering assignment quality. This yields values that are slightly smaller than 1.0 for those methods.
For decreasing gap sizes r_O, where sequences are not yet fully overlapping, the F1 score decreases successively for all approaches. This is due to the fact that elements of patterns become increasingly assigned wrong patterns, which creates segments that contain patterns of different content than the ground truth patterns. In LTS this effect is very drastic, as the initial run for determination of active regions already yields long clusters, which are potentially combinations of multiple ground truth patterns. This results in bad initial cluster and pattern assignments. For range segmentation shrinking gap sizes increase the probability that multiple adjacent patterns are assigned the same pattern. However, as in our experiment the range is chosen close to the ground truth, this effect occurs late. Up until a ratio of $r_O = 0.2$ it is well able to reconstruct the true patterns and clusters. The window approaches show a similar effect, as a window size that is close to the ground truth yields a good segmentation, where less pattern elements end up in a wrong window. With increasing overlap, the number of such misassigned pattern elements increases yielding worse accuracy.
For the case of fully overlapping ground truth patterns, all approaches are decreasing significantly in performance. This results from the increasing dominance of the effects that are discussed above. Moreover, a perfect overlap of $r_O = 1.0$ makes it impossible to distinguish patterns due to the following reasons. When assuming a high number of observations all pairwise combinations of patterns occur subsequently in the data. Such combined patterns again form patterns for themselves. Thus, apart from temporal characteristics (i.e. time gaps between pattern elements), no indicator exists that separates both patterns. In current approaches this type of temporal aspects is not included, which needs to be improved in the future to solve this problem.

Effect of the Total Number of Samples / Scalability: The scalability of approaches is investigated by varying the parameter n_{dat}, which yields increasingly large test traces. Performance is measured in terms of assignment quality with results shown in Figure 6.5 and in terms of run time with measurements shown in Figure 6.6.
Parameters: $K = 10$, $r_O = -1.5$, $r_C = 0.2$, ($\mu_{len} = 10$, $\sigma^2_{len} = 2$), ($\mu_{seq} = 10$, $\sigma^2_{seq} = 0.5$) and ($\mu_{\Delta t} = 6, \sigma^2_{\Delta t} = 0.5$). In particular the overlap ratio was chosen such that the gap is close enough that the transition between two consecutive sequences is blurred.

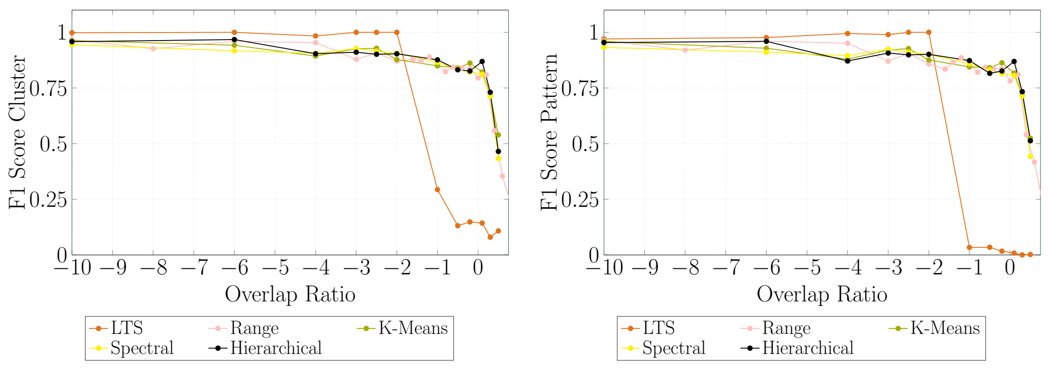


Figure 6.4: The F1 score of clustering and pattern assignments is presented, when overlap is increased.

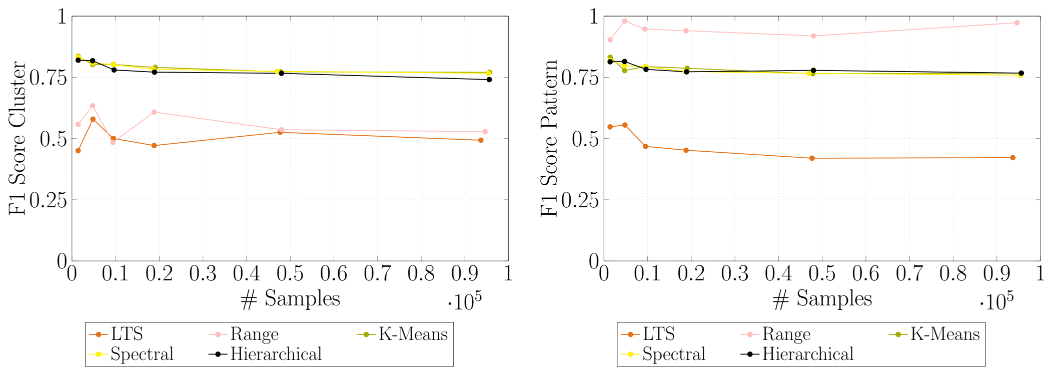


Figure 6.5: The F1 score of clustering and pattern assignments is presented, when the length of the trace is increased.

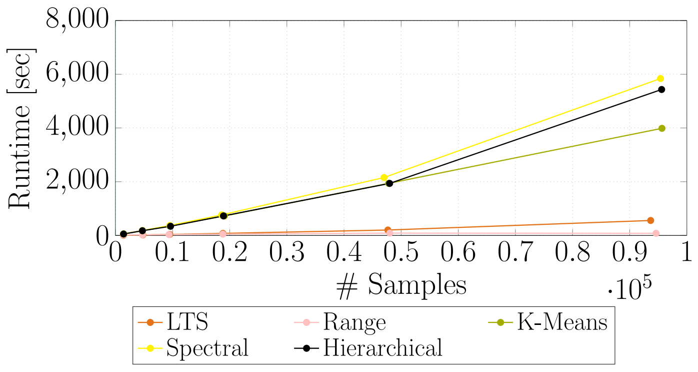


Figure 6.6: The run times are shown, that result when the length of the trace is increased.

Results: As Figure 6.5 shows the F1 score starts at different points. This is because r_O is chosen in a range were consecutive sequences are potentially overlapping which decreases performance as was investigated above. Furthermore, for an increasing number of samples the F1 score remains constant for clustering and pattern assignment. Thus, the number of samples does not have an influence on assignment quality. This makes sense, as segmentation is identical, no matter how long the overall trace is. This produces identical pattern candidates that are clustered (which in number are less for shorter traces and more for longer ones).
In terms of run time, the number of samples does have an effect as Figure 6.6 shows. It can be seen that range segmentation performs best among all approaches, as it is the most lightweight method. At the same time it allows for good scalability as both detection of ranges and vectorization can easily be parallelized. LTS shows good performance as well, due to the following reason. After the initial cluster assignment the complexity of the problem is broken down, as one only operates on active and meta clusters. Above that, the complexity of merging and combining clusters is low in terms of computational effort. In contrast to LTS and range segmentation, the window-based approaches perform worse with increasing n_{dat}. The problem is, that the initial search for a suitable number of windows is computationally expensive. In the given case the search space was within a range of 50 windows around the true size.

Further Evaluation: In this section a synthetic evaluation is performed to show the characteristics and performance of the discussed approaches. Application of this segmentation clustering in a real life scenario is extensively evaluated in the case study given in Chapter 9.

6.7 Summary and Conclusion

Based on a subset of relevant TVs, segments and cluster assignments need to be discovered from an inspected trace. In general in MSSs such segments might be overlapping. However, within the scope of this work non-overlapping elements are assumed.
In this section several approaches were extended and compared in terms of suitability to be used for this task. Those include window-based and range-based approaches as

well as an extended version of LTS. It was found that for clear gaps all approaches are equally well suited if parameterized correctly. However, for decreasing gap sizes range segmentation and window-based approaches work best. In particular, range segmentation scales best in terms of run time, which is why this approach is used within the proposed DM pipeline.
Lastly, refinement clustering is discussed as an optional step within the DM pipeline, which allows to include expert input to find clusters of appropriate granularity.

7 Modeling Multivariate State Sequences

In the previous chapters a procedure is presented that allows to identify functional procedures and its relevant TVs, yielding multiple sets of MSSs that each represent recordings from executions of a certain functional procedure. Thus, each set contains one relevant aspect of system behavior which can be used to understand the system functioning and that can be used to extract specifications from it. However, for this the functional procedure has to be characterized and represented in state and time, for each set of MSSs, such that knowledge is extracted from those.
In this chapter suitable probabilistic models for this are presented and assessed in terms of applicability for mining of specifications and for representation of MSSs.
This forms the penultimate step of the proposed DM pipeline. Formally in this step an input MSS $M_i \in \hat{M}$, a model type T, a learning approach L and the hyper parameters for learning P_L are given. In particular T is a probabilistic model and the learning approach L can be split into a structure discovery and a parameter estimation approach. A learned model Q is output. The operation performed is

$$Q = learn(M_i, T, L, P_L) \tag{7.1}$$

Both potential models and its learning approaches are discussed in this chapter. In particular this includes TSCBNs which are optimized for the representation of MSSs.

Modeling MSSs: Functional procedures consist of MSSs at this point. As presented in Chapter 2 such MSSs are represented in terms of TVs and state sequences of TVs. Also, three important models can be used in different ways for its representation as it is shown in Figure 7.1. As described in [8] three TVs - *grass*, *sprinkler* and *rain* - are shown. Dynamic evolution of those TVs is modeled as state sequences that capture the state of a TV at any point in time, e.g. in Figure 7.1 the TV rain starts in the *no* and changes to the *yes* state. Thus, the state sequence of this TV can be described as $\langle no, yes \rangle$. State sequences of multiple TVs are called MSSs. In MSSs states of TVs may be causally dependent on each other, e.g. if the TV grass is in the *dry* state, the TV sprinkler being at state *yes*, will change the TV grass' state into *wet*. Thus, the grass' state depends on the sprinkler's state in this stage of the process.

Challenges and Comparison: Modeling such MSSs is challenging due to multiple reasons

- *Latent State Changes:* In real world data per TV only *state changes* are observed. That is, for the case where an interval with a certain state is followed by an interval with the same state, the transition between those intervals is not observed, while it still might have a causal influence on other states. In the example of Figure 7.1 the TV grass has the true (but unobserved) state sequence $\langle dry, wet, wet \rangle$, as once

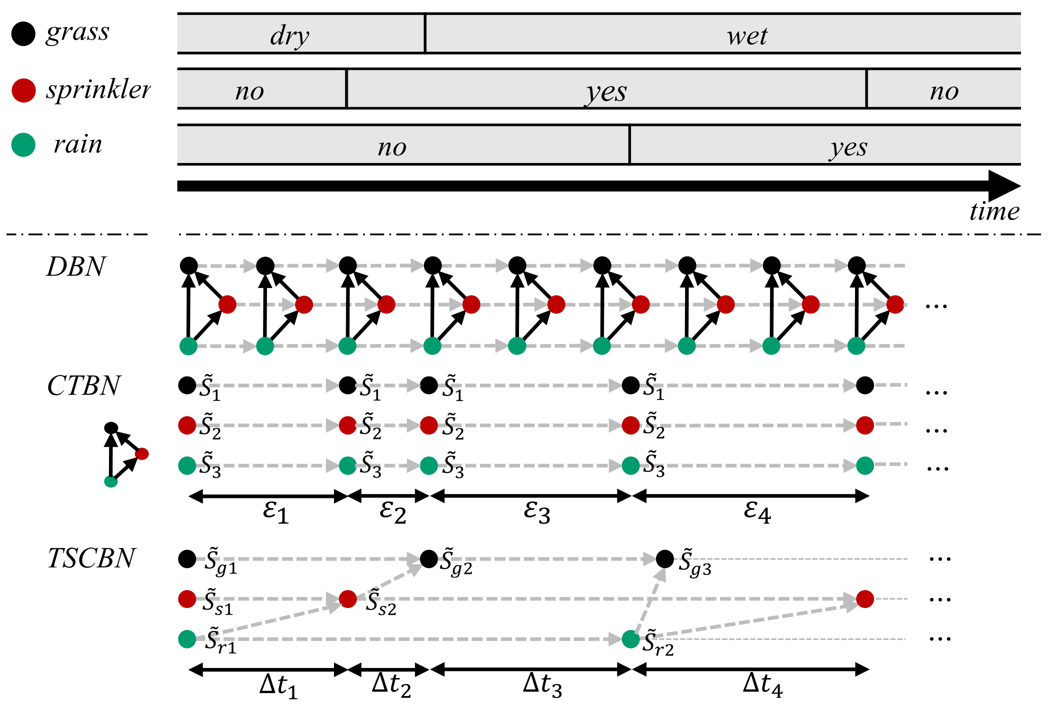


Figure 7.1: An MSS with 3 Temporal Variables for the process of wetting grass, and models to generate it, are shown. If the grass is dry the sprinkler turns on, if it is not raining. Once it starts raining the sprinkler turns off. The same model could produce a sequence where the rain is falling throughout the process. Then, the sprinkler would never have turned on. Temporal State Change Bayesian Networks provide a compact yet expressive representation for such scenarios.

the rain starts falling it has a causal influence on the TV grass' state by making it wet. However, what is actually observed in this case are only the state changes $\langle dry, wet \rangle$. Thus, the actual sequence $\langle dry, wet, wet \rangle$ is *latent*. Estimating this latent sequence from what was observed is challenging as multiple valid sequences could have been produced (e.g. $\langle dry, dry, wet \rangle$). Both, Discrete Bayesian Networks (DBNs) and Continuous Time Bayesian Networks (CTBNs) do not explicitly model such state changes, but contain this information implicitly. In contrast to that TSCBNs do explicitly model such state changes and approximate those at locations where latency is observed only.

- *Parameter Complexity:* Another challenge lies in the number of parameters that are required for modeling MSSs. With growing width and height (i.e. number of TVs) of the network an increasing number of parameters is required to expressively represent the MSS. CTBNs store the transition matrix of the network and intensity matrices per condition of its structure for this purpose, with this, keeping parameter complexity low. DBNs need to repeatedly store the causal structure across discrete time steps introducing parameters at each discretization step. With growing number of steps (i.e. with growing precision) this results in an exploding parameter space for such networks. TSCBNs model events or state changes only at points in time where those occur, which keeps the parameter space low, when modeling MSSs.

- *Interpretability:* To allow expert input and understand system behavior it is essential to provide a representation that is interpretable. In general this means that the structure captures behavior in an understandable manner and no overhead in network complexity is present, i.e. network complexity is reduced to the essential core of the modeled behavior. DBNs are interpretable in the sense that the static structure within and across time-slices allows to understand correlations between TVs and which TVs influence which over time. However, this structure is static and thus, does not model the full functional procedure in an interpretable way. CTBNs represent the total behavior in matrix representations. This allows to interpret consequent behavior of certain TVs based on a set of conditions of other TVs. The complex interplay of those matrices however, does not directly allow to interpret the functional procedure underlying the set of MSSs. TSCBNs model procedures in terms of events at its points in time where a state change is possible. Although such events might be latent it allows to understand the set of possible interval structures in an intuitive manner and the causal influences between those in an interpretable manner.

- *Expressiveness:* While CTBNs require less parameters, this comes at the cost of reduced expressiveness when modeling processes of MSSs. That is, as CTBNs generalize over the process by using identical matrices throughout the process, which does not distinguish causal correlations that might change over time. DBNs discretize behavior over time. With this, changing dependencies are also not represented explicitly but need to be modeled in terms of an extended inter time slice structure. This introduces an increased degree of data fragmentation throughout each time slice, which again requires a high degree of approximation at each step with this making it less expressive. TSCBNs model each state change explicitly which allows to capture defined functional procedures of MSSs in the way they are correlated. With this data fragmentation is kept low and correlating edges are reduced to relevant ones only making those models well express such procedures.

- *Finiteness of Processes:* CTBNs store matrices of intensities that are used to determine the consequent state based on a current state of TVs. This can be repeated arbitrarily often, which allows to represent procedures of infinite length. In contrast to that, both DBNs and TSCBNs define finite processes, as when trained procedures of finite length result in structures of finite length. However, infinite procedures could be theoretically modeled when chaining multiple models, such that the output state of a finite model forms the input model of another model.

Chapter Outline: First, as TSCBNs are based on BNs and its theory, background on this topic is given in Section 7.1. Next, TSCBNs and a novel structure discovery approach are presented. To put those into context the state of the art in those fields is described in Section 7.2. In this Section also an overview of the three models is given, before in Section 7.3 TSCBNs, its structure discovery approaches and parameter estimation methods are presented as introduced in [8]. Lastly, in Section 7.4 the evaluation is given, in order to show the applicability of TSCBNs for the modeling of MSSs in the context of Specification Mining. The overall chapter is based on [8].

7.1 Background

7.1.1 Bayesian Networks

BNs are Directed Acyclic Graphs (DAGs) that represent causal relationships between Random Variables (RVs) X_i as nodes and their dependencies as edges. Such networks define a Joint Probability Distribution (JPD) $P(X_1, X_2, ..., X_n)$ over all n RVs in the graph, which allows for various types of probabilistic inference tasks, including diagnosis or prediction. JPDs are computed from Conditional Probability Distributions (CPDs) $P(X_i|Pa(X_i))$ which are defined per RV X_i in the network conditioned on its respective parents $Pa(X_i)$. For a set of variables $X_1, X_2, ..., X_n$ the JPD is computed with

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i)). \tag{7.2}$$

To infer probability distributions along the network under given evidence, exact approaches, such as marginalization of JPDs or Bayes' theorem can be used.
Mostly, for ease of computation of JPDs, RVs are assumed to be either all of discrete or all of continuous type. If both types occur in the network simultaneously, the graph is called a Hybrid Bayesian Network (HBN). In HBNs the set of RVs X is divided into two sets $X = Y \cup Z$, where Y represents discrete and Z continuous RVs. In this work the notion of HBNs is restricted to the case, where continuous RVs can only be conditioned on discrete ones and the inverted case is not allowed. A common way for inference in such cases is to assume fixed distributions (e.g. Gaussian or Exponential) defined with

parameters Θ per continuous node Z and to condition each Θ on Z's discrete parents, e.g. assuming one discrete parent Y, each parent outcome y gives a distribution Σ for the respective node. That is,

$$Z|Y = y \mapsto \Sigma(\Theta(y)). \tag{7.3}$$

7.1.2 Conditional Independence

Independence: With X and Y as RVs with value spaces Val(X) and Val(Y). Given a Probability distribution P, X and Y are independent if

$$P(X = x, Y = x) = P(X = x) \cdot P(Y = y) \tag{7.4}$$

for all $x \in$Val(X) and $y \in$Val(Y). Independence between RVs is denoted as $X \perp Y$.

Conditional Independence: Given are a set of additional RVs $\mathbf{Z}$ with values Val($\mathbf{Z}$ with values $\mathbf{z} \in$Val($\mathbf{Z}$. With this, X is conditionally independent of Y given $\mathbf{Z}$ if

$$P(X = x, Y = x|\mathbf{Z} = \mathbf{z}) = P(X = x|\mathbf{Z} = \mathbf{z}) \cdot P(Y = y|\mathbf{Z} = \mathbf{z}) \tag{7.5}$$

for all $x \in$Val(X), $y \in$Val(Y) and $\mathbf{z} \in$Val($\mathbf{Z}$). Conditional independence between RVs is denoted as $(X \perp Y)|\mathbf{Z}$.

Conditional Independence in BNs: The BN $\mathcal{G}$ and the set $NonDescendants_{\mathcal{G}}(X_i)$ of nodes that are not descendants of X_i in $\mathcal{G}$ is assumed, i.e. there is no path in $\mathcal{G}$ from X_i to any node in $NonDescendants_{\mathcal{G}}(X_i)$. With this, BNs are defined such that $\mathcal{G}$, for each node X_i and all nodes $X_j \in NonDescendants_{\mathcal{G}}(X_i)$, encodes the conditional independences

$$(X_i \perp X_j)|Pa_{\mathcal{G}(X_i)} \tag{7.6}$$

That is, any RV in the network is conditionally independent of its non-descendants given its parents.

Markov Blanket: The Markov Blanket $\mathcal{B}(X_i) \subset \mathbf{X}n\{X_i\}$ is the minimal set of nodes, that for all of its nodes $X_j \notin \mathcal{B}(X_i)$ satisfies the conditional independence

$$(X_i \perp X_j)|\mathcal{B}(X_i) \tag{7.7}$$

In a BN this includes all parents, children and Co parents of X_i.

7.2 State of the Art

As this work proposes both a model and a dynamic Structure Discovery (SD) algorithm related works in those fields are presented. Related work on parameter estimation is omitted as, no novel estimation approach is presented, but rather extensions to the classical approaches Expectation Maximization (EM), Variational Inference (VI) and Maximum Likelihood Estimation (MLE) are discussed.

7.2.1 Temporal Probabilistic Models

In past research several models for temporal reasoning under uncertainty were introduced.

Time-sliced BNs: The first group of models includes time-sliced BNs such as discrete time nets [192] with static structures and its dynamically adjusting extensions [194, 195, 196, 197] and DBNs, that break the Markovian assumption [198] or the stationarity assumption [199]. Main caveats of such networks include its static structure and its high overhead due to repetition of structures.

Interval-based BNs: The group of interval-based BNs models temporal events or intervals as states of nodes. A subgroup of those BNs defines the type of event as state, and the interval in which the event occurred as outcome, in a node. There nodes represent irreversible events whose outcomes are a cross product of discretized intervals and state outcomes of events [200, 201]. But, for higher precision this leads to huge outcome spaces and events are irreversible. Other models represent a certain event only, with times of occurrence as outcome [202]. Temporal Bayesian Network of Events (TBNE) that were introduced in [203] are comparable to TSCBNs. Similar to TSCBNs, there, intervals are modeled as tuples of time of occurrence of a state change and states of the next interval including the option of remaining in the same state. However, TBNEs omit the following aspects, which are solved in TSCBNs. First, interval lengths are discretized which allows for less temporal precision. Second, no notion for representation of dynamic causal relations between states of different TVs is defined. Third, the model does not allow to model causal dependence of state transitions that remain in the same state, which is solved in TSCBNs using latent sequences. Networks, such as Modifiable Temporal Belief Networks (MTBN) [204] allow to model intervals in terms of nodes for start, end and duration of an interval. There again each node represents the time and duration of one event only. In [205] the modeling of temporal information is accomplished by using HBNs consisting of discrete nodes for the values and continuous nodes for the time delays. However, rather than modeling dynamics of a process as TSCBNs do, those model dependence between time and state statically.

Markov Process based models: To model continuous time under uncertainty Markov Processes and Continuous-Time Markov Processes are used. In [193], Nodelmann introduced CTBNs. This model stores a generalized parameter set over all TVs under defined conditions, while TSCBNs use one parameter set per temporal occurrence of an event. In [206] Tawfik considered events as distributions over time. This approach unlike us does not model defined processes of intervals, but rather static causal dependencies of continuously evolving processes.

General Probabilistic Networks: Apart from BNs in Causal Probabilistic Networks [207] event occurrence and its times of occurrences are modeled separately. That means, dynamic causal influence of states on duration of intervals of TVs is not included. In [208] Probabilistic Temporal Interval Networks nodes are temporal intervals and edges are uncertain interval relations modeled in terms of Allen's relations. Thus, intervals are considered rather qualitatively (e.g. as *duration*, *before*) than quantitatively.

Specialized Models: Further, in many domains specialized models were proposed. In reliability engineering object oriented BNs [209, 210], Markov Chains, Boolean Logic Driven Markov Processes [211], Dynamic Fault Trees [212] and Event Sequence Diagrams [213] were applied for temporal modeling under uncertainty. Those approaches focus on object interactions and fault detection without general extensibility and applicability to interval modeling. Models of temporally evolving dependencies further include the approach proposed in [185], where evolving Markov Random Field based models are used for time-series clustering or in [214] and [215], where temporal evolution in social networks is modeled. However, the further is optimized for time-series, while the latter is focused on graph data rather than MSSs.
Consequently, existing solutions for modeling MSSs, result in complex representations both in terms structure and parameter size.

7.2.2 Structure Discovery Approaches

SD approaches in BNs are typically categorized in score-based, constraint-based and hybrid approaches. When it comes to discovery of dynamic structures, further the categories of learning of temporal BNs and PM are used, which is a related field for learning process models from sets of sequences.
The learning of structures of BNs is a challenging problem due to the exponential increase of the space of potential network structures with the number of nodes. Most approaches reduce complexity by constraining the space of allowed structures, specifying orders of variables, applying temporal conditions, including prior knowledge or limiting the network size.

Score-based: Such approaches start with an initial graph, evaluate the graph using a score function and try to ultimately optimize the structure to maximize the function's score. Performance of those methods depends on the choice of reasonable scoring functions, well constraining the space of allowed structures, by designing an appropriate optimizer and by encoding of the networks. Common scores include the Bayesian Information Criterion (BIC) [216] or the Minimum Description Length (MDL) [217]. Classical approaches include the K2 algorithm [218] or Greedy Hill Climbing (GHC). Those approaches do not guarantee to converge to the global optimum. For global optimization this was solved in several ways. Those include among many others formulating a linear optimization problem and solving it with branch and cut [219], by using shortest-path search [220], constraint programming [221], genetic algorithms [222], simulated annealing [223] or particle swarm optimization [224]. Score-based approaches work well for less data, but suffer from bad computational performance for growing data. In addition to that, directions of edges are not clear, which is especially relevant when modeling temporal data.

Constraint-based: Such approaches constrain potential edges and use Conditional Independence (CI) tests to find the network structure. Typical CI tests include e.g. the χ^2 or G tests. Traditional approaches include the SGS approach [225] or the PC algorithm [226], which use CI tests on subsets of RVs to discover structures. Possible constraints include structural constraints given by properties of the specific objective network [227] (e.g. Markovian assumption in DBNs) or heuristics to limit potential parents of nodes [228, 229]. This type of methods tends to find hidden common causes, handle selection bias and work well with sparse graphs [230].

Hybrid: Other works connect score and constraint-based approaches to combine their best properties. This was done by CI tests to find initial ordering of RVs or initial graph skeletons, which are used as input for consequent optimization [231, 232, 233].

Temporal BN learning: When learning static DBNs the Markovian assumption is assumed and the discovery is decomposed into learning edges within a time slice and between subsequent time-slices. In [234] the further is done with above approaches as in the static case, while the latter reduces to a feature selection problem where each node chooses one or more parents from the previous time-slices. In further works, SD approaches for DBNs with no Markovian assumption [198] or stationarity assumption [235, 199], were presented. Further approaches use event sequences as input and Process Discovery algorithms to create the network. The approach described in [236], extracts a DAG from event sequences by adding edges to the BN if two events directly follow each other in multiple event sequences. Then, in case of cycles in the model dedicated edges are removed. However, this approach is not applicable to the given scenario, as here the aim is to model different events of the same TV using more than one node. This strategy was improved by the authors in [237]. There, a unique label is assigned to each of the events to prevent the formation of cycles in the BN and the labeling of events is done in a naive way. That is, all the i-th occurrences of an event of TV X are labeled X_i. The algorithm that is presented in this chapter finds an advanced solution to this problem of uniquely labeling events, in that it allows to handle optionally occurring events. That is, e.g. in sequences $\langle\ B,\ A,\ A,\ B\ \rangle$ and $\langle\ B,\ A,\ B\ \rangle$, in the first sequence labeling of A might be A_1 or A_2. The approach in [238] uses the Heuristic Miner [23] to create a BN, which is a Process Discovery algorithm that creates a dependency graph between events based on the frequencies of specific observations. There, cycles in the dependency graph are resolved by using dummy nodes.
All the algorithms in this section so far assumed dependencies between events if the events directly follow each other in the event sequences. No statistical tests or score optimization is used to check for independencies. In [239] the authors use a score-based procedure to filter the edges in the dependency graph of the Heuristic Miner using a mutual information score. Nevertheless, there still remains the problem that edges in the TSCBNs are allowed that connect events that do not directly follow each other in the event sequences. The algorithm that is presented in this chapter drops this restriction.

Process Discovery: In the PM community multiple algorithms were proposed for mining a process model from event logs (usually extracted from business processes). The algorithms proposed in this field are related to the discovery of BNs, when time is

included. However, those approaches do not model multiple dimensions and TV structure but rather event sequences. Important algorithms of this type include the Alpha Miner [240] for discovery of Petri nets, which does not allow for silent transitions (i.e. optionally occurring events in sequences are not represented). This was extended in the Heuristic Miner [241] that includes support of edges in terms of occurrence frequencies, but cannot handle the concept of parallel events. This was improved in the Inductive Miner [242] and Inductive Miner infrequent [242], where sequences and its representations are decomposed according to defined rules that allow to extract multiple temporal concepts (e.g. parallel, subsequent).
All of the aforementioned approaches are not optimally suited for the effective SD of TSCBNs from observed MSSs. This is, first, as the concept of dimensions of TVs with interactions between TVs in state and time is not inherently represented, which however can be exploited to reduce the amount of candidate structures. That is, edges between subsequent nodes of the same TV are implicitly given in TSCBNs and only edges between state changes of two different TVs within defined time ranges are relevant. Moreover, optional occurrences of state changes are not handled and the event-to-node assignment is not solved in an optimal manner, as stated above. Therefore, in Section 7.3.4 TrieDiscover is introduced that solves those challenges for learning TSCBNs from sets of MSSs. This approach can be categorized as hybrid, as it uses temporal and structural constraints extracted from control and time flow information extracted from event sequences. This reduces the space of allowed structures to a subspace of temporally meaningful structures and allows to define directions of edges between RVs in the direction of correct order time flow. By consequently applying Constraint-Based (CB) or Score-Based (SB) approaches on the remaining candidates, valid BNs of good precision are discovered with a reasonable performance, which will be shown in Section 7.4.

7.2.3 Bayesian Models for Temporal Data

7.2.3.1 Dynamic Bayesian Networks

Dynamic Bayesian Networks were introduced in [192] and are in its core regular BNs, but with a defined structure. Those networks model stochastic processes in state and time by slicing the temporal space into slices. Per time-slice a static BN is used to define correlations between RVs within each slice, i.e. at certain times. Each RV in a time-slice corresponds to the state of a TV at that time. To model dynamics, RVs between time-slices are connected with edges in the direction of time flow, where RVs that correspond to the same TV have the same symbol indexed with the number corresponding to the according time-slice. Each time-slice has a fixed structure that links it to one of the d next time-slices. The maximum forward reach of an edge to a successive time-slice is referred to the order of the DBN. Formally, this is defined as follows.

Given a set of RVs $\mathbf{X} = \{X_1, ...X_n\}$, with observation X_1^τ at time slice τ all observations of RVs at that time are written as $\mathbf{X} = \{X_1^\tau, ...X_n^\tau\}$. A DBN of order $d = 1$ can be divided into two BNs $(B_0, B_\mapsto)$, where B_0 defines the initial probability distributions $P(\mathbf{X}^0)$ and $B_\mapsto$ is a temporal BN that includes two time-slices defining the probability

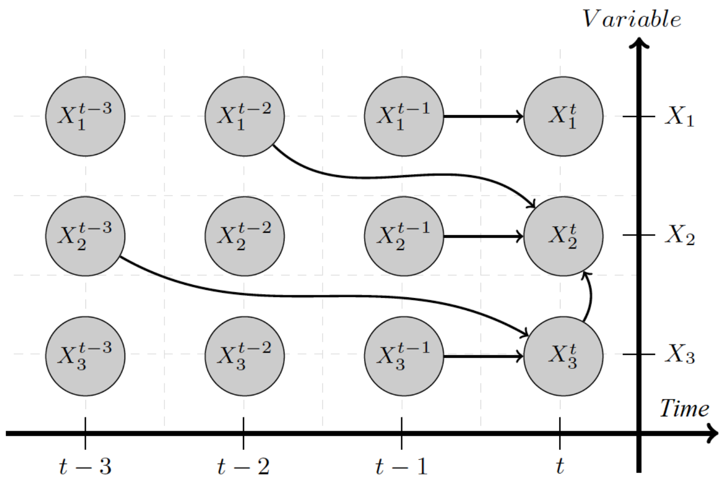


Figure 7.2: A DBN of order 3 is shown here.

distribution $P(\mathbf{X}^t|\mathbf{X}^{t-1})$. In this case the JPD of the DBN is expressed as

$$P(\mathbf{X}_{0:T}) = \prod_{t=0}^{T}\prod_{i=1}^{n} P(X_i^t|Pa(X_i^t)) \tag{7.8}$$

, where $\mathbf{X}_{0:T} = \{\mathbf{X}^0, \mathbf{X}^1, ..., \mathbf{X}^T\}$. For the general case of a DBN of order d B_0 is defined identically to the above case, while $B_{\mapsto}$ is defined with $d+1$ time slices $t-d, ...t$ with probability distribution $P(X_i^t|Pa(X_i^t))$ per time slice t, where the parent set $Pa(X_i^t)$ can contain nodes $X_i^\tau, i \neq j$ for $\tau \in \{t-d, ..., t-1, t\}$, nodes X_i^τ for $\tau \in \{t-d, ..., t-2, t-1\}$ or may be empty.
That is, as can be seen in Figure 7.2, in $B_{\mapsto}$ nodes in time slice t have only incoming edges that originate from a RV of the same TV in time slices of $[t-d, t-1]$ or from a RV of a different TV in time slices of $[t-d, t]$.
Within the scope of this work DBNs of first order are considered only.

7.2.3.2 Continuous Time Bayesian Networks

In [193] Nodelman introduced CTBNs which are essentially a combination of Markov Processes, whose interactions are defined in a BN. All relevant terms are introduced in the following.

Markov Processes of TVs: According to [193] such processes are defined as matrices of transition intensities. Each entry (i, j) gives the intensity for transitioning from state i to state j. Here, homogeneous Markov Processes are used as a basis, i.e. processes where the transition intensities do not depend on time.
The TV X has the domain $Val(X) = \{x_1, x_2, ...x_n\}$ which gives the homogeneous

Markov Process $X(t)$ for this TV which is defined via its intensity matrix

$$\mathbf{Q}_X = \begin{bmatrix} -q_1^x & q_{12}^x & \cdots & q_{1n}^x \\ q_{21}^x & -q_2^x & \cdots & q_{2n}^x \\ \cdots & \cdots & \cdots & \cdots \\ q_{n1}^x & q_{n2}^x & \cdots & -q_n^x \end{bmatrix} \tag{7.9}$$

where $q_i^x = \sum_{i \neq j} q_{ij}^x$. The intensity q_i^x is the probability of leaving state x_i and the intensity q_{ij}^x the probability of transitioning from x_i to x_j.
Starting from a state $X(0) = x_i$ the TV stays in that state for a fixed amount of time defined by an exponential distribution of parameter q_i^x which gives a PDF for x_i remaining in state x_i as

$$f(t) = q_i^x \exp(-q_i^x t) 0 \leq t \tag{7.10}$$

When transitioning X changes to state x_j with probability q_{ij}^x / q_i^x.

Conditional Markov Processes of TVs: According to [193] this is a type of in homogeneous Markov Process where intensities vary over time, but not as a function of time, but rather the intensities are a function of the current values of a set of other TVs. For a TV (=RV) Y that is conditioned on a set V of TVs (=RVs) the conditional intensity matrix (CIM) are defined as.

$$\mathbf{Q}_{Y|V} = \begin{bmatrix} -q_1^y(V) & q_{12}^y(V) & \cdots & q_{1m}^y(V) \\ q_{21}^y(V) & -q_2^y(V) & \cdots & q_{2m}^y(V) \\ \cdots & \cdots & \cdots & \cdots \\ q_{m1}^y(V) & q_{m2}^y(V) & \cdots & -q_m^y(V) \end{bmatrix} \tag{7.11}$$

Example: For a TV $E(t)$ that defines over time if a person is eating or not is expressed with TV hungry $H(t)$ which can be in states h_1 = not hungry or h_1 = hungry. Intensity matrices

$$\mathbf{Q}_{E|h_1} = \begin{bmatrix} -0.01 & 0.01 \\ 10 & -10 \end{bmatrix} \mathbf{Q}_{E|h_2} = \begin{bmatrix} -2 & 2 \\ 0.01 & -0.01 \end{bmatrix} \tag{7.12}$$

imply that if a person is hungry he will begin eating in 1/2 hour, while a person who is not hungry and eating to stop eating in 1/10 hour.

Continuous Time Bayesian Networks: According to [7] a CTBN models a stochastic process over a structured state space consisting of assignments to a set of local variables (i.e. TVs in the context of this work $\mathbf{X} = \{X_1, X_2, ...X_k\}$. Dynamics of those variables is modeled per TV as a Markov Process that is conditioned on a set of other variables $\mathbf{U}$. With this as defined in [7], a CTBN N over $\mathbf{X}$ consists of two components, an initial distribution $\mathbf{P}_{\mathbf{X}}^0$ specified as a BN B over $\mathbf{X}$ and a continuous transition model, specified as

- a directed graph G whose nodes are $X_1, \ldots, X_k$, where $Pa_G(X_i) = U_i$ denotes the parents of X_i in G.

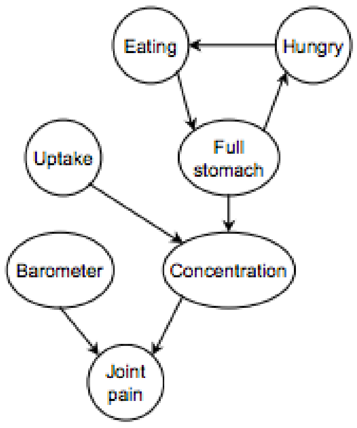


Figure 7.3: Here an example of a CTBN structure is shown [7]. At each time step a Markov Process that is conditioned according to this structure models the temporal evolution of the data.

- a conditional intensity matrix $\mathbf{Q}_{X_i|U_i}$ for each variable $X_i \in \mathbf{X}$

Example: The example used in [7] is shown in Figure 7.3. It describes the process of taking a drug to alleviate pain, e.g. the concentration of the drug in the blood depends on how full the stomach of the patient is. Also, it shows that whether a person is hungry depends on how full the stomach is and how it is affected after eating. Thus, it stores the transition structure (i.e. the network) as well as the intensity matrices.

7.3 Temporal State Change Bayesian Networks

7.3.1 Model

In [8], the TSCBN model was formally and descriptively introduced. Here the formalization is used in Section 7.3.2. Also, the explanation of the model is used to define the state structure in Section 7.3.3, the temporal structure of TSCBNs in Section 7.3.3.1 and the compact representation of the model in Section 7.3.3.2.

7.3.2 Formal Definition

7.3.2.1 Model

Let's assume a set $\mathcal{S}$ of TVs S_i that each temporally evolves over a set of states $\Xi_i = \{s_{i1}, s_{i2}, ...\}$, which can be dynamically interdependent.

Definition 7.3.1. TSCBN A *TSCBN* is a HBN $B = (G, \Theta)$ that consists of a parameter set Θ and a DAG $G = (N, E)$ of nodes N connected via directed edges E.

Definition 7.3.2. Nodes In B, N forms the set of nodes $N = V \cup T$ which each represent an occurrence of a state change $* \mapsto s_{ij}$ of a TV S_i from its previous state $*$

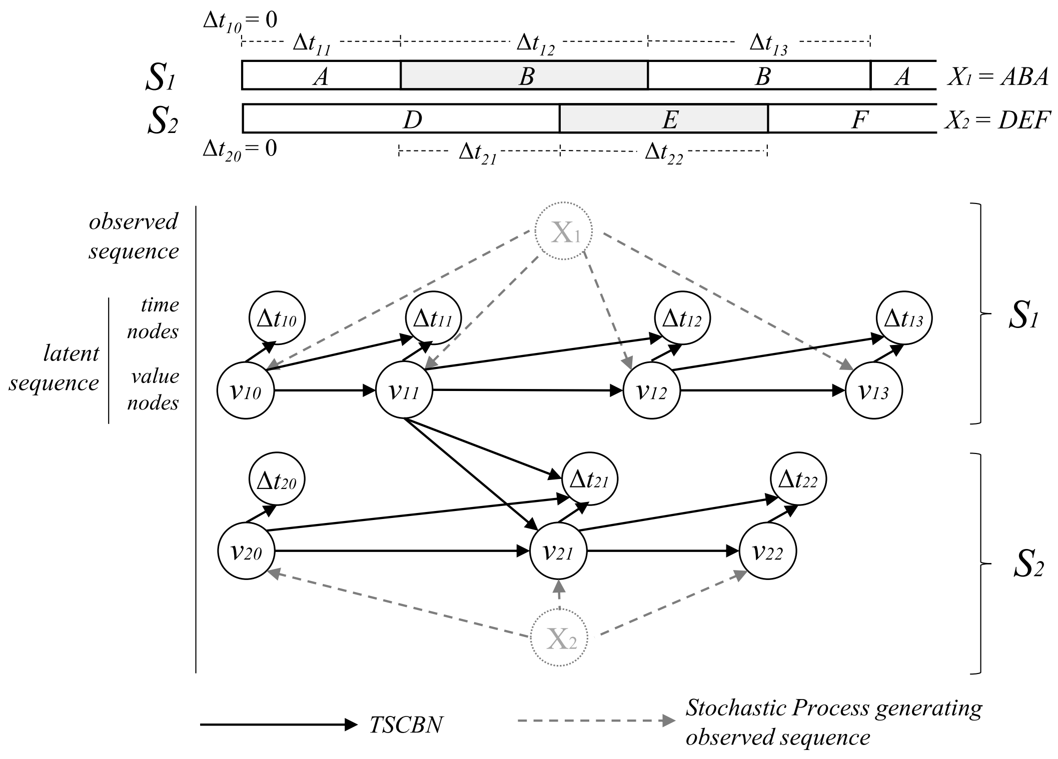


Figure 7.4: Two TVs S_1 and S_2 are illustrated. The top part shows a MSS that is *generated* by the TSCBN shown in the lower part. X_1 and X_2 indicate the *observed sequence* of state changes, which are generated by the true *latent sequences* $\langle(v_{10}, \Delta t_{10}), (v_{11}, \Delta t_{11}), (v_{12}, \Delta t_{12}), (v_{13}, \Delta t_{13})\rangle$ and $\langle\ (v_{20}, \Delta t_{20}), (v_{21}, \Delta t_{21}),\ (v_{22}, \Delta t_{22})\rangle$. Note that a temporal-causal dependency between state change v_{11} of TV S_1 and state change v_{21} of TV S_2 is given in the shown TSCBN [8].

to its next state s_{ij}. This occurrence is defined by state nodes $v_{ik} \in V$ and temporal $\Delta t_{ik} \in T$.

Definition 7.3.3. State Nodes All *state nodes* v_{ik} are discrete RVs that indicate the state s_{ij} a TV changes to. V forms the set of all state change nodes in the network.

Definition 7.3.4. Temporal Nodes Each *temporal node* Δt_{ik} is a continuous RV that defines the relative temporal distance to its latest occurring predecessor. T forms the set of all temporal nodes in the network.

Definition 7.3.5. Initial nodes All temporal nodes S_i have an initial node $n_{i0} = (v_{i0}, \Delta t_{i0}) = (s_{idef}, 0)$ at time $t_{abs}^{i0} = 0$ in a default state $s_{idef} \in \Xi_i$.

Definition 7.3.6. Edges An edge in E indicates a causal dependency between connected state changes. Edges between nodes of the same TV (*intra-variable*) and edges that define dependencies among TVs (*inter-variable*) are distinguished here.

Definition 7.3.7. Intra-variable edges It is assumed that each consecutive state change depends on its preceding state. The corresponding edges are defined as *intra-variable edges.* That is, for m TVs the following holds:

$$(v_{i(k-1)}, v_{ik}) \in E, \forall i \in [1, m], \forall k > 0 \tag{7.13}$$

$$(v_{i(k-1)}, \Delta t_{ik}) \in E, \forall i \in [1, m], \forall k > 0 \tag{7.14}$$

Further, the temporal behavior Δt_{ik} depends on the state v_{ik} a TV changes to, which is implied by the edge

$$(v_{ik}, \Delta t_{ik}) \in E, \forall i \in [1, m], \forall k \leq 0 \tag{7.15}$$

Definition 7.3.8. Inter-variable edges If a state change $n_{iq} = (v_{iq}, \Delta t_{iq})$ of TV S_i causally depends on a change n_{kr} of another TV S_k an edge defined as *inter-variable edge* results with

$$(v_{iq}, v_{kr}) \in E, i \neq k \tag{7.16}$$

$$(v_{iq}, \Delta t_{kr}) \in E, i \neq k \tag{7.17}$$

Definition 7.3.9. State model The set of all state nodes and its edges forms the *state model* of a TSCBN. According to the defined HBN structure, those state nodes v of a TSCBNs are a network of discrete RVs.

This definition allows inference algorithms of standard BNs to be applied on the state structure of a TSCBN. This is done regardless of the states' times of occurrence as temporal nodes are leaf nodes, e.g. the Most Probable Explanation (MPE) could be determined to identify dominant system behavior. Also, the JPD for the state part of a TSCBN with m temporal values, with n states per node could be simply computed with

$$P(v_{10}, v_{11}, ...v_{mn}) = \prod_{i=1}^{m} \prod_{k=0}^{n} P(v_{ik} | Pa(v_{ik})). \tag{7.18}$$

Definition 7.3.10. Temporal model The set of all temporal nodes and its edges forms the *temporal model* of a TSCBN. There, each node Δt is a continuous RV, with a distribution Σ, defined by parameters Θ. All Δt_{ij} are conditionally dependent on the same parents $Pa(v_{ij})$ of states v_{ij} and the state v_{ij} of the node itself. As all Δt are conditioned solely on discrete parents, the temporal part of each state change can be written as

$$\Delta t_{ij} | Pa(v_{ij}) \cup \{v_{ij}\} \mapsto \Sigma(\Theta(Pa(v_{ij}) \cup \{v_{ij}\})). \tag{7.19}$$

Definition 7.3.11. Absolute time TSCBNs represent time relatively. Thus, the absolute time t_{abs} of a state change event needs to be determined from its latest parent's absolute time. If the event of state change was not observed at a parent node, the time of a state node is measured relatively to its last occurring TV state change $\bar{Pa}$. This can be expressed as

$$t^{ik}_{abs} = \max_{r,s}(\bar{Pa}(n_{rs}).t_{abs}) + \Delta t \tag{7.20}$$

where $\bar{Pa}(n_{rs})$ are all parents of v_{rs}, that did occur, $\max\limits_{r,s}(\bar{Pa}(n_{rs}).t_{abs})$ indicates the latest occurring TV node and Δt is the temporal gap from this parent node to the absolute time of the current node.

7.3.3 Modeling State Sequences

Univariate State Sequence First, let's only consider the state sequence of TV S_1 in Figure 7.4, which is modeled by the upper part of the TSCBN (i.e. nodes v_{1k}). In MSSs only state changes are observed and a TV has a sequence of states it is in at any point in time. If a state change occurs, this state change is observed in the data, e.g. $X_1 = \langle A, B, A \rangle$ is observed, where each sequence element indicates the state a TV changes to at a certain point in time. However, if a TV has two consequent intervals with identical states, this change is not measured and thus, cannot be observed in the data, e.g. in S_1 the actual latent state sequence is $\langle A, B, B, A \rangle$. But, as the second occurrence of B is not observed multiple latent sequences could be assumed from this observation, e.g. the latent sequence also could have been $\langle A, A, B, A \rangle$ or $\langle A, B, A, A \rangle$. As causal dependencies in MSSs actually depend on the true latent occurrences, TSCBNs model each (potentially latent) state change of the latent sequence as state node v_{ij}. Those v_{ij} are modeled as discrete RVs, e.g. the example sequence of S_1 shown in Figure 7.4 corresponds to the outcomes $(v_{10} = A, v_{11} = B, v_{12} = B, v_{13} = A)$ of the TSCBN. These latent outcomes produce the observed sequence $X_1 = \langle A, B, A \rangle$.
When estimating the latent parameters v_{ij} of a TSCBN from observed sequence examples of X_i, only valid latent sequences (i.e. the ones that can produce an observed sequence) should be considered. An estimation approach to this is presented in Section 7.3.6.
Further, each consecutive state change depends on its preceding state. Thus, in TSCBNs an edge between all consecutive state nodes of the same TV (e.g. v_{10} to v_{11}) is defined, e.g. for the TV S_{move} in Figure 7.5, the state change to *run*, is less likely when the person was *sitting* than when it is already *walking*.

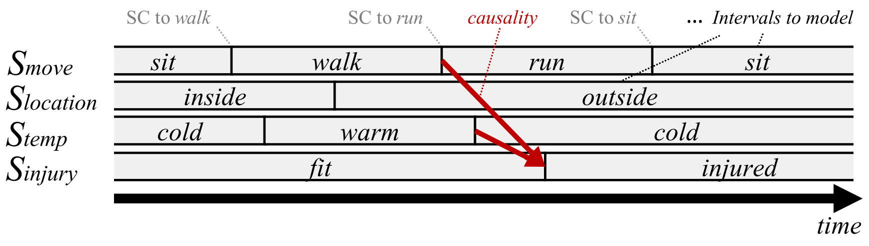


Figure 7.5: A MSS of the movement of a person is shown, with four dependent TVs S_{move}, $S_{location}$, S_{temp}, S_{injury}. The MSS is $M =< (0, 4, S_{move}, sit), (0, 5, S_{temp}, cold), ... >$. Also, the main concept of TSCBNs is shown. Intervals are modeled by SCs. Causal dependence between intervals (*here:* S_{move}*,* S_{temp} *and* S_{injury}) is modeled as edges between SCs to respective intervals [8].

Multivariate State Sequence So far only one TV was considered. If multiple TVs are given, states of TVs may causally depend on states of other TVs. Such causal dependence is defined with an edge between the corresponding states' changes, e.g. in Figure 7.4 the state of v_{21} the TV S_2 changes to, not only depends on its previous state (as defined above), but also on the state change v_{11} of TV S_1. Also, in Figure 7.5, if the TV S_{temp} changes its state to being *cold*, this may more likely cause the consequent state *injured* of the TV S_{injury} (together with the previous state of the same TV, which is *fit*).

7.3.3.1 Modeling Times of Sequences

While the state nodes v_{ij} capture the state a TV changes to, those nodes do not capture the time at which this state change occurs. To solve this, in TSCBNs each state node v_{ij} is connected to a temporal node Δt_{ij} that defines its time. This node is modeled as a continuous RV. The outcome of this RV is the relative time gap to its last predecessor. That is, if a state node has exactly one predecessor node, this gap is computed relatively to the absolute time of this predecessor nodes' state change, e.g. if v_{12} occurred at time 17 and the outcome of its consequent state change v_{13} is $\Delta t_{13} = 3$, the time at which the state change v_{13} occurred is 20 $(= 17 + 3)$.
Further, temporal nodes Δt_{ij} are conditionally dependent on the same parents $Pa(v_{ij})$ as its corresponding state nodes v_{ij} and on the node v_{ij} itself, e.g. in Figure 7.5, the time it takes to get *injured* depends on when the person started *running* and when it got *cold*.
Consequently, if a state node has two or more predecessor nodes (i.e. parent nodes) the absolute time of this node's state change is determined relatively to the time of the parent nodes' state change that occurred last, e.g. in Figure 7.4 the absolute time of the state change v_{21} is determined relatively to the absolute time of its parents v_{11} and v_{20}. If change v_{11} occurred at time 7, v_{20} at time 0 and $\Delta t_{21} = 4$, the time of v_{21} is determined relatively to v_{11} (as $7 > 0$). The resulting absolute time of the state change v_{21} would be 11 $(= 7 + 4)$ in this example.
As all times of state changes are defined relatively to their parents, all TVs S_i require a

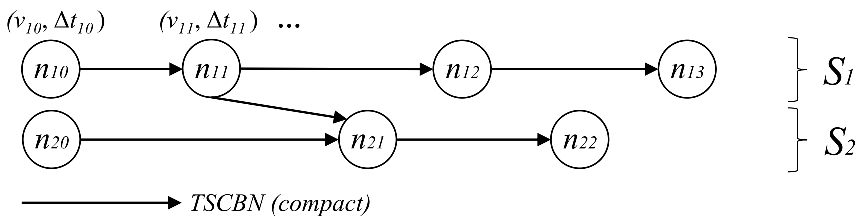


Figure 7.6: The compact representation of a TSCBN defines each node n as the state a TV S changes to and its time of change [8].

reference point at an initial default state at absolute time $t_{abs}^{i0} = 0$, i.e. $\Delta t_{i0} = 0$.

Absolute time in latent sequences: As stated above in a latent sequence nodes may not be observed and thus, may have no absolute time, e.g. in S_1 the time of transition from B to B is not observed. Thus, using this point as reference for succeeding nodes' time of occurrence Δt is not possible. In this case the relative time is measured from the point of the last observed state change. If for a node in a TV no parent occurred, the last observed state change of the corresponding TV is used as reference point, which is at worst the initial state. This initial state is well defined at time $t_{abs}^{i0} = 0$ for each TV. For instance, let's assume the outcome $\langle B, A, B, A \rangle$ for sequence S_1 with observed absolute times of state change $\langle 0, 5, 8, 10 \rangle$ and $\langle D, D, F \rangle$ for sequence S_2 with observed times $\langle 0, 7 \rangle$. Then, the time of $v_{21} = D$ is not observed. Thus, no absolute time of this nodes is known from observation. That is, the absolute time of the state change $v_{22} = F$ cannot be found relatively to v_{21}. Instead this hidden node v_{21} is skipped and the next observed parent v_{20} is used as reference in a TSCBN. Thus, in this example $\Delta t_{22} = 7$ (under the given condition of state changes) defines the absolute time of v_{22} to be 7 $(= 0 + 7)$.
Moreover, in this TSCBN structure it is assumed that the duration of a state does not influence the consequent state and the duration of a TV, i.e. all Δt are leaf nodes in the HBN structure, which enables tractable inference as the discrete state structure is independent of continuous nodes. Instead continuous nodes are conditioned on discrete states, which can easily be expressed as distribution parameters, which are conditioned on the parent outcomes.

7.3.3.2 Compact Representation

In practice it is often required to discuss such models with experts (e.g. for specification mining as described in Section 8.4.2). For this and for the purpose of a simpler visualization the TSCBN can be written in a more compact manner. Each state change v_{ik} and its time of change Δt_{ik} can be condensed to one node $n_{ik} = (v_{ik}, \Delta t_{ik})$. Also, connecting edges between two nodes v transform to connecting edges between the corresponding nodes n. The corresponding edges to temporal nodes Δt are implicitly assumed, e.g. the model of Figure 7.4 can be condensed to the model shown in Figure 7.6.

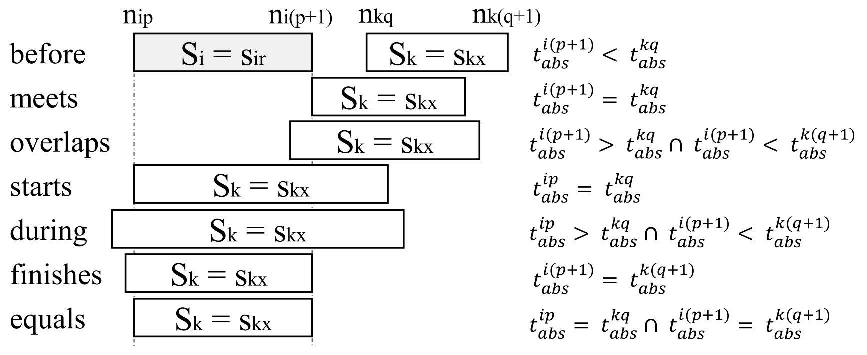

nip
ni(p+1)
nkq
nk(q+1)
before
meets
overlaps
starts
during
finishes
equals
Si = sir
Sk = skx
$t_{abs}^{i(p+1)} < t_{abs}^{kq}$
$t_{abs}^{i(p+1)} = t_{abs}^{kq}$
$t_{abs}^{i(p+1)} > t_{abs}^{kq} \cap t_{abs}^{i(p+1)} < t_{abs}^{k(q+1)}$
$t_{abs}^{ip} = t_{abs}^{kq}$
$t_{abs}^{ip} > t_{abs}^{kq} \cap t_{abs}^{i(p+1)} < t_{abs}^{k(q+1)}$
$t_{abs}^{i(p+1)} = t_{abs}^{k(q+1)}$
$t_{abs}^{ip} = t_{abs}^{kq} \cap t_{abs}^{i(p+1)} = t_{abs}^{k(q+1)}$

Figure 7.7: Allen's temporal relations are shown on the left, between a TV S_i that is in state s_{ir} for a certain time interval and a TV S_k that is in time interval s_{kx}. Each interval boundary is modeled by a node n of a TSCBN. On the right the according temporal requirements for the absolute times (defined by the respective Δt) of each node n is illustrated [8].

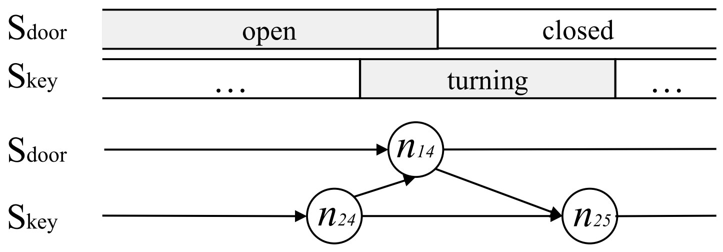


Figure 7.8: An example of the overlaps relationship, which shows the state of the door and the key as TV. When the key is *turned*, the *open* door changes to a *closed* state. The overlaps relationship is modeled as TSCBN, with v_{14} as state change from *open* to *closed* and with v_{24} as start and v_{25} as end of the key *turning* procedure [8].

7.3.3.3 Properties

TSCBNs have multiple properties that make it well suited for modeling MSSs, which is revised here shortly. First, TSCBNs compactly represent temporal-causal behavior with JPDs and due to its structure allow for straight forward application of BN inference algorithms. Second, nodes can be defined as intervals of infinitesimally small length (i.e. Δt close to zero). This allows to model dependence between events and intervals by using TSCBNs. Third, TSCBNs allow to model all basic temporal interval concepts as proposed by Allen, which is shown in Figure 7.7 and Figure 7.8. Fourth, as time nodes are leaf nodes, they can each be modeled with different distributions which allows to model realistic variations in the temporal behavior. Lastly, all states are defined in time as they are recursively computed from its predecessors.

7.3.4 TrieDiscover

To allow for inference, the structure of TSCBNs needs to be discovered from a set of observed MSSs. For this in [8] TrieDiscover was introduced, which allows to learn an

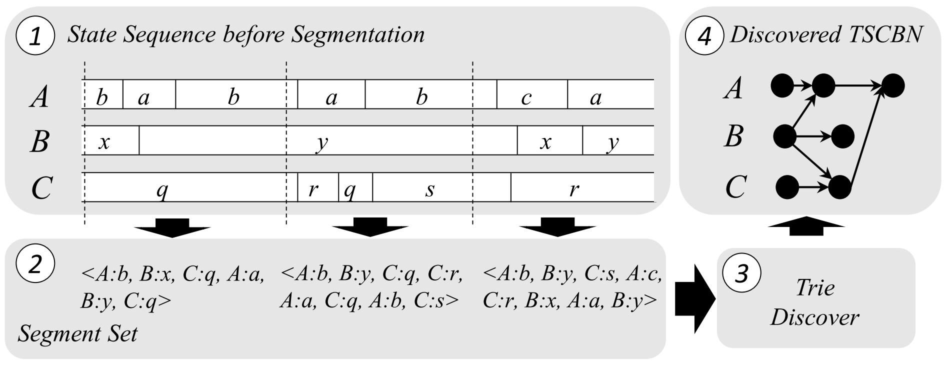

1 State Sequence before Segmentation
A b a b a b c a
B x y x y
C q r q s r
2 <A:b, B:x, C:q, A:a, B:y, C:q> <A:b, B:y, C:q, C:r, A:a, C:q, A:b, C:s> <A:b, B:y, C:s, A:c, C:r, B:x, A:a, B:y>
Segment Set
3 Trie Discover
4 Discovered TSCBN
A
B
C

Figure 7.9: The basics step of the discovery approach are shown. Starting from a long trace the input MSSs for TrieDiscover are deduced by segmentation. With this TrieDiscover finds a BN structure to represent a set of MSSs [8].

interpretable structure which is used for Specification Mining. As this approach is used in the proposed DM pipeline its basic steps are introduced in the following.

Challenges: Learning TSCBNs from MSSs is challenging due to many reasons, First, as with increasing numbers of TVs and lengths of MSSs, the number of network structures grows. Further, due to data fragmentation a high number of observed examples is needed. Outliers need to be filtered to reduce noise which is often present in real world and values might be missing, i.e. MSSs do not necessarily contain values for all nodes of a TV in the TSCBN. Lastly, to precisely infer the network structure all events of an observed MSS need to be assigned to a node of a TV sequence in the TSCBN. This is not trivial. Especially, for the case where the number of observed events per RV in a MSS succeeds the number of nodes for that RV in the TSCBN. For example given three nodes for TV B and a sequence $\langle A, B, A \rangle$ the observed B might be mapped on either B_0, B_1 or B_2. A naive assignment would assign the i-th occurrence of B to the i-th node B_i. However, possibly this event could have been influenced by the preceding event A_1. For that case an event-assignment to B_2 could have been more suitable.
Those challenges are solved in *TrieDiscover* as the number of structures and the number of samples is restricted by aggregating paths in tries, outliers are filtered out by defining suitable thresholds, missing values are averaged and events are assigned appropriate nodes by using a defined merging strategy on branches of the trie. With this TrieDiscover is well suited for learning of functional procedures from MSSs.

As exemplified in Figure 7.9 this approach takes multiple MSSs as an input and produces a TSCBN structure, that represents the variations of functional procedures, as output. Notably, input sequences are segmented such that an initial state at time zero is contained. If that state is not given a default state is defined per TV. It consists of six steps, which are presented in the following.

Assumptions: To be able to reconstruct the structure as TSCBN a sufficient numbers of occurrences of correlating events need to be observed. Further, correlation is assumed

if two TVs are dependent. With this, it is assumed that sufficiently identical variations of functional procedure executions are observed, i.e. the set of MSSs contains sufficient observations of correlating state changes of TVs. Especially for the case of Specification Mining this needs to be given, as then, dominant states are preserved, noise is filtered out and interpretability of the structure is given.
For the case of functional procedures of high diversity a TSCBN can still be discovered with TrieDiscover, but will result in more edges and dependence on the given states of TVs at each stage. This is, as the model capacity of TSCBNs is higher with increasing number of edges, while in this case neither dominant states nor Specifications are extracted. This is due to the Markov assumption which per definition allows BNs to only represent conditions one step after and before each node. However, if it is assumed that functional procedures are sufficiently similar, as described in the first case, each path can be seen as recorded under a defined set of conditions per node, thus, being unique per path that a functional procedure can take. With this, the dominant system behavior as well as the extraction of Specification in terms of paths is assumed. Thus, within the scope of the proposed DM pipeline it is assumed that both clustering and segmentation resulted in MSS sets of sufficient similarity and thus, all learned models each represent a corresponding functional procedure and its variations.

7.3.4.1 TrieDiscover Steps

In the following all of the above steps are presented using the running example of [8]. This input sequence could have been produced by any of the segmentations presented in Chapter 6 and thus, can be seen as MSSs of multiple executions of a functional procedure that is to be modeled.

Input set of MSSs: The example MSSs consist of sequences of TVs and its state changes. As described above at first only a sequence of the TVs without its states are considered. Those sequences are recorded from a functional procedure with less variations. Thus, a set of TVs $\mathcal{S} = \{A, B, C, D, E, F, G\}$ is assumed that produced the following executions.

$$\begin{aligned}\mathcal{M}_{obs} = \{&\langle A, B, A, D, A, E, G\rangle, \langle A, C, A, D, A, F, G\rangle,\\ &\langle A, B, D, A, E, G\rangle, \langle A, C, D, A, F, G\rangle\}\end{aligned}$$

At first in steps 1 to 5, i.e. during Parent Set identification, per node temporal information is exploited to determine a set of candidate parents per node.

1. Trie Creation: In this step the set of TV sequences $\mathcal{M}_{obs}$ is compressed into a trie by merging similar prefixes, such that any path through the trie corresponds to an observed one dimensional sequence of TVs. For the example this gives the following.

$$\begin{tikzcd}
 & & A \arrow[r] & D \arrow[r] & A \arrow[r] & E \arrow[r] & G \\
 & B \arrow[ur] \arrow[dr] & & & & & \\
 & & D \arrow[r] & A \arrow[r] & E \arrow[r] & G & \\
A \arrow[uur] \arrow[ddr] & & & & & & \\
 & & A \arrow[r] & D \arrow[r] & A \arrow[r] & F \arrow[r] & G \\
 & C \arrow[ur] \arrow[dr] & & & & & \\
 & & D \arrow[r] & A \arrow[r] & F \arrow[r] & G &
\end{tikzcd}$$

Figure 7.10: A Trie modeling the observation of four MSSs is shown [8].

2. Filtering: At each edge within the trie the occurrence frequency of the according transition is observed in $\mathcal{M}_{obs}$. Based on a threshold infrequent transitions are considered as noise and dropped. For this in TrieDiscover the parameter k is defined. It defines that if an outgoing edge e of a node X occurs less than k times the total number of observations of node X, i.e. $frequency(e) < k \cdot frequency(X)$, then it is filtered.

3. Subtree Merging: To join identical sequences, sub-paths within the network are merged as shown in Figure 7.11. Based on theory from automata theory. For this TrieDiscover converts the graph into an automata equivalent, by translating edges in a trie to states and nodes to state transitions that were triggered by an event. By using the minimization approach of [243], which has linear complexity $\mathcal{O}(n)$, the structure is compressed further. Basically, this approach assigns a sub-tree code to each node in the structure and stores it in a hash map that maps the sub-tree code to a hash value. If two nodes are assigned the same sub-tree code, the nodes are merged, which yields maximal lossless compression. For the example sequence this gives the following.

$$\begin{tikzcd}
 & & A \arrow[dr] & & & & \\
 & B \arrow[ur] \arrow[rr] & & D \arrow[r] & A \arrow[r] & E \arrow[dr] & \\
A \arrow[ur] \arrow[ddr] & & & & & & G \\
 & & A \arrow[dr] & & & & \\
 & C \arrow[ur] \arrow[rr] & & D \arrow[r] & A \arrow[r] & F \arrow[uur] &
\end{tikzcd}$$

Figure 7.11: The DAWG after minimizing the trie in Figure 7.10 is given [8].

4. Event-to-Node Assignment: At this point the structure contains variants of observed sequences. However, this does not include the assignment of events to nodes, i.e. each node in the graph needs to be assigned an index in the TSCBN. TrieDiscover does this in two steps, by traversing the graph once in topological (= Event Assignment) and once in reversed (= Event Refinement) topological order.

- *Event Assignment:* In an initial assignment round starting from the root node all preceding nodes in the TSCBN are passed to each node in the graph by traversing it. For instance, this parent set at the root node is { A_0, B_0, C_0, D_0, E_0, F_0, G_0 }.

At each node the index of the node's TV is increased by one and assigned to the graphs node. In case of multiple parent nodes (e.g. node G in Figure 7.11) the parent set that is passed from either parent is merged. For the above graph this yields

$$\begin{gathered}
\langle A_1,\, B_1,\, A_2,\, D_1,\, A_3,\, E_1,\, G_1\rangle \\
\langle A_1,\, C_1,\, A_2,\, D_1,\, A_3,\, F_1,\, G_1\rangle \\
\langle A_1,\, B_1,\, D_1,\, A_3,\, E_1,\, G_1\rangle \\
\langle A_1,\, C_1,\, D_1,\, A_3,\, F_1,\, G_1\rangle.
\end{gathered}$$

- *Event Refinement:* This assignment might be ambiguous, as there might be an event gap between two events on a path in the graph such as in the following.

$$\begin{tikzcd}
 & \mathrm{B}_1 \arrow[r] & \mathrm{C}_1 \arrow[r] & \mathrm{B}_2 \arrow[r] & \mathrm{D}_1 \arrow[dr] & & \\
\mathrm{A}_1 \arrow[ur] \arrow[dr] & & & & & \mathrm{F}_1 \arrow[r] & \mathrm{B}_3 \\
 & \mathrm{C}_1 \arrow[r] & \mathrm{B}_? \arrow[rr] & & \mathrm{E}_1 \arrow[ur] & &
\end{tikzcd}$$

Here the lower path contains one B as opposed to two in the upper path. Thus, both B_1 or B_2 are valid assignments. By backwards passing the parent set to its nodes per node missing elements (e.g. B_2 on the lower path) are identified. To determine which assignment to use, the number of strongly connected components (SCC) is compared. An SCC results when event nodes in the DAWG that are assigned to the same TSCBN node, are merged. Assigning B to B_1 results in

$$\begin{tikzcd}
 & \mathrm{B}_1 \arrow[r] \arrow[d, bend left] & \mathrm{B}_2 \arrow[r] & \mathrm{D}_1 \arrow[dr] & & \\
\mathrm{A}_1 \arrow[ur] \arrow[dr] & & & & \mathrm{F}_1 \arrow[r] & \mathrm{B}_3 \\
 & \mathrm{C}_1 \arrow[u, bend left] \arrow[rr] & & \mathrm{E}_1 \arrow[ur] & &
\end{tikzcd}$$

, where the number of SCCs is seven here, as B_1 and C_1 form one SCC. Assigning C_1 to B_2 gives 8 SCCs and thus, is preferred resulting in this structure.

$$\begin{tikzcd}
 & \mathrm{B}_1 \arrow[dr] & & & \mathrm{D}_1 \arrow[dr] & & \\
\mathrm{A}_1 \arrow[ur] \arrow[rr] & & \mathrm{C}_1 \arrow[r] & \mathrm{B}_2 \arrow[ur] \arrow[dr] & & \mathrm{F}_1 \arrow[r] & \mathrm{B}_3 \\
 & & & & \mathrm{E}_1 \arrow[ur] & &
\end{tikzcd}$$

In case of a change in assignment the potential parents of the successor nodes need to be updated.

5. Parent Candidate Identification: So far each node was assigned a parent set, while merged nodes might have different parent sets before merging, e.g. D which had A_1 and B_1 or A_1 and C_1 as parents. As both scenarios are possible, both C_1 and B_1 are kept in the parent candidate set of D_1.

Further, two events X_i and Y_j might occur in parallel (i.e. sometimes event X_i precedes Y_j and sometimes Y_j precedes X_i) resulting in both nodes containing one another in its parent set. If this is the case, it is assumed that the two events do not influences each other, i.e. are independent, and thus, are removed from both sets. Those events are identified at points where the graph contains SCCs with more than one node (e.g. as above in the cycle between B_1 and C_1).
Lastly, the parent set is further reduced by including a temporal threshold t_{th}. Per edge a average time delay between corresponding events is computed and for edges where the time delay exceeds a threshold t_{th}, the preceding node is removed from the potential parents of the succeeding node.

6. Structure Optimization: To determine parents within the graph that are correlating and thus, form an edge in the TSCBN, structure discovery approaches of static BNs are used to reduce the set of candidate parents per node. This is done either using score optimization procedures or using CI tests as described in Section 7.2. TrieDiscover was introduced in three variants, that differ in this step.

1. *Score-based (SB) TrieDiscover* uses a decomposable score (e.g. BIC, AIC, K2, BDeu) for optimization [220].
2. *Constraint-based (CB) TrieDiscover* uses CI tests to find the optimal parent set by removing edges with a significance level lower than a threshold α.
3. *Extended constraint-based (CBv) TrieDiscover* performs a χ^2 test to filter out connections between RVs (i.e. nodes of TVs) that are below a correlation threshold χ_{th} and successively applies CI tests similar to CB TrieDiscover.

In our implementations sbTD used the exact score optimization approach *parent graph* introduced in [220] for Structure Optimization. This approach works as follows. Parent graph calculates the score for small parent sets first. If a parent set is non-optimal, also parent sets that include the non-optimal parent set are non-optimal. A heuristic is used to decide for which parent set the next score is calculated.

Conversion to TSCBN: The resulting structure can now be transformed to a TSCBN. Intra-edges are defined inherently by connecting nodes that correspond to the same TV, e.g. B_0, B_1 and B_2. Inter-edges for each node of a TV are defined by the parent set of that node that remains after the structure optimization step.

7.3.5 Discussion of TrieDiscover

As the trie representation allows to capture a large part of structural variations and filtering can be adjusted, a structure with high information content can be generated even for paths that are observed less frequently. Correlation tests require a sufficient amount of observations to assess correlation. With this, rules need to be defined for the case of subsequent events that rarely occur. In this case it is possible to either consider such transitions as noise, to include those or to let an expert assess it. Especially for the case of Specification Mining the latter two scenarios are preferred as even rare occurrences, but with a high likelihood might indicate nominal behavior. Further, the

granularity of the approach can be well adjusted by defining temporal, as well as score- and CI test thresholds. With this, the degree of information loss can be controlled as a hyper parameter.
With this, TrieDiscover yields a both interpretable, lossless and expressive structure that captures correlations of the functional procedures among multiple TVs in state and time.

7.3.6 Parameter Estimation

With the structure learned, parameterization of a TSCBN needs to be performed based on a set of given MSSs. In [8] it was introduced that for fully observed MSSs (i.e. if all state changes are observed are if state changes are events) classical approaches for parameter estimation in BNs can be used, which is Maximum Likelihood Estimation or Bayesian Inference. For the general latent case where MSSs are not fully observed three approaches where presented. Those are a sampling based MLE approach, Expectation Maximization and Variational Inference. In the following those approaches are revised, as those are used in the proposed DM pipeline for parameter estimation of TSCBNs. For this the same input MSSs as in the case of SD are used.

7.3.6.1 Assumptions

Latency: Per TV, observations are never fully observed, which requires to approximate each observation along multiple nodes, e.g. in Figure 7.9 for TV S_1, $\langle A, B\rangle$ of one TV correlates to a subset of nodes n_{11}, n_{12}, n_{13} in the TSCBN, where nodes of subsets might be interdependent via inter-edges. Further, the mapping per observation to possible latent estimates is constrained in time, i.e. temporal order of events needs to conform with nodes and directions of edges in the TSCBN, and in state, i.e. per TV only a combinatorial subset of outcomes is possible.

Formalization: Each latent sequence of a TV S_i is defined as a latent variable $Z_i = (v_{i1}, v_{i2}, ...)$ and the observed sequence as X_i. The set of all latent variables for observation k is $Z^k = \{Z_1^k, Z_2^k, ..., Z_m^k\}$ and the set of all observed sequences is $X^k = \{X_1^k, X_2^k, ..., X_m^k\}$. $X = \{X^1, X^2, ...\}$ are all observations and $Z = \{Z^1, Z^2, ...\}$ the according latent sequences. For the latent case all Z^k are assumed missing at random and the empirical distribution, that results from the observations Z are denoted as $q(v_{ij}|Pa(v_{ij}))$.

Challenges: Challenges include first, to guarantee consistency of partially observed MSSs with the TSCBN. That is, temporal consistency and valid combinatorial mappings of state sequences need to be ensured. Second, ambiguity of nodes needs to be handled, i.e. the structure of a TSCBN may require more nodes per TV than were observed. In this case the problem occurs that assigning k observed state changes to n nodes is ambiguous, e.g. for $n = 5$ and an observed sequence $\langle A,\ B,\ A,\ C\rangle$ the actual state changes could be $\langle A,\ A,\ B,\ A,\ C\rangle$, $\langle A,\ B,\ B,\ A,\ C\rangle$, $\langle A,\ B,\ A,\ A,\ C\rangle$ or $\langle A,\ B,\ A,\ C,\ C\rangle$.

Discussion: For the task of specification mining exact approaches are preferable if fully observed MSSs are given. With this high likelihood is only given if it was observed.

However, in practice, especially in the testing and development phase that is assumed here, often data is noisy and incomplete. In this case approximate approaches are required. This results in approximates of the estimated likelihoods. Such approaches tend to still maximize the likelihood of the part of the data that was observed, i.e. if partial aspects of a MSS are fully observed, that part in the data is still more likely than its approximates. With this assumption even the approximate approaches are still valid to use in the proposed DM pipeline for learning of TSCBNs for specification mining.

7.3.6.2 Non-latent Estimation: MLE and Bayesian Inference

In this case it is assumed that all state changes of the input sequence were observed. In this case MLE or Bayesian Inference can be used, which is revised in the following based on [244] .

Counting and MLE: The goal of MLE is to set the parameters Θ such that those maximize the likelihood of the data. These ML estimates Θ^* of the state structure of a TSCBN can be computed independently under the i.i.d. assumption. For this, according to [244] it is possible to decompose the network structure, such that per node v_{ij} and parent outcome set $Pa(v_{ij}) = t$ the following holds for each CPD

$$p(v_{ij} = s | Pa(v_{ij}) = t) \propto \sum_{n=0}^{N} \mathbb{I}[v_{ij}^n = s | Pa(v_{ij}^n) = t]. \tag{7.21}$$

This corresponds to the fact, that the CPD entry $p(v_{ij} = s|Pa(v_{ij}) = t)$ can be set by counting the number of times $\{v_{ij} = s|Pa(v_{ij}) = t]\}$ was seen in the data set Z for fixed joint parental state t [244]. In this scenario counting and MLE is identical.
For bigger networks data fragmentation becomes problematic as for higher numbers of parameters more samples need to be observed. This also means that few samples for parameter estimation tend to overfit the data [245]. Unobserved CPDs are set to be uniform in the given scenario to allow for inference. Further, fragmentation decreases with less parent nodes and discrete parent values. TSCBNs are designed to minimize the further, while the latter needs to be restricted.

Bayesian Approach: According to [244] in this approach each node is governed by a parameter which is assumed to have a distribution itself, with the JPD of the network as $p(z_1, z_2, ...)$, which can be represented with its parameters as

$$\begin{aligned} p(z_1, z_2, ...) &= p(z_1|Pa^*(z_1); \Theta_{z1})p(z_2|Pa^*(z_2); \Theta_{z2})... && (7.22)\\ &= \Theta_{z1}^{Pa^*(z_1)} \Theta_{z2}^{Pa^*(z_2)}... && (7.23)\end{aligned}$$

, where z_i are all nodes v_{ij} of the TSCBN and $Pa^*(z_i)$ is a defined conditional set of z_i's parents. Assuming global parameter independence and observations $\mathbf{X}$ the posterior is represented as

$$p(\Theta_{z1}, \Theta_{z2}, ...|\mathbf{X}) \propto p(\Theta_{z1}|x_1^*)p(\Theta_{z2}|x_2^*)... \tag{7.24}$$

, where Θ_{zi} is the product of all CPDs of its node $\Theta_{zi}^{Pa^*(z_i)}$ and x_i^* the observation at node i. This allows to learn parameter posteriors separately. As each parameter Θ_{zi} is still multidimensional it is common to further assume local parameter independence, which is that Θ_{zi} is the product of all CPDs of its node $\Theta_{zi}^{Pa^*(z_i)}$. With this the posterior both factorizes over parental states and the local parameters, which means that

$$p(\Theta_{zi}|x_i^*) \propto p(x_i^*|\Theta_{zi})p(\Theta_{zi}^{Pa^*(z_i)_1})p(\Theta_{zi}^{Pa^*(z_i)_2})... \tag{7.25}$$

By assuming i.i.d. data and the local and global parameter prior independencies, Dirichlet priors can be used to now determine the parameters. This is explained in more detail in according literature [244].

Temporal Nodes: Temporal nodes are simply estimated by fitting Gaussian distributions over each node under the respective conditions.

7.3.6.3 Approximate Maximum Likelihood Estimation via Randomization

As latent parts of a set of MSSs are never observed, estimates need to be found to approximate the parameters of the TSCBN. In [8] we introduced a standard sampling based approximate MLE approach (MLE-R) for this. There continuous temporal nodes are estimated separately from state nodes, which can be done as those are leaf nodes.

Temporal Node Estimation: As temporal nodes depend on its corresponding state node, as well as on this nodes' parents, one distribution needs to be found per condition combination. For this, first, per node Δt and per condition, a distribution is defined (e.g. Gaussian). Second, during the MCMC sampling procedure all n_{samp} gaps for Δt are recorded and stored under its respective parent conditions. Then, per condition set at each node Δt, the recorded gaps are used to fit a distribution over the observed gaps and the corresponding distribution parameters are used in the temporal nodes.

State Node Estimation: Latency is handled by imputing missing values [246] at each observation and then applying non-latent approaches (e.g. MLE) for parameter estimation.
In TSCBNs the structure is well known and all possible mappings from short observed sequences to long latent sequences are known. Thus, in this approach for each observed TV sequence X_i^k a valid mapping to the full sequence Z_i^k is found through randomly sampling from a parameterized TSCBN. With this parameter estimation of the TSCBN is performed as follows.

1. Per input sequence X_i^k draw a valid random mapping from the current TSCBN to get Z_i^k. This results in a list of full observations Z (in state and time).
2. Use Z to perform MLE as described in Section 7.3.6.2 to find parameters of the TSCBN.
3. Repeat step 1) and 2) until convergence, with the newly updated parameters that were found in 2).

7.3.6.4 Expectation Maximization

The following EM approach is proposed in [8].

EM Formalization: In EM the goal is to maximize the likelihood $L(\Theta|X, Z)$ of the TSCBN given the observed data. This means, it maximizes the expectation of *(i)* the TSCBN with parameters Θ to produce all latent sequences Z and *(ii)* Z and Θ to produce the observed sequences X. Formally, this means EM maximizes

$$\Theta = \arg\max \mathbb{E}_{Z \sim p(Z|X,\Theta)}[\log p(X, Z|\Theta)]. \tag{7.26}$$

Temporal Constraints: To improve accuracy of the EM algorithm (when estimating the latent Z^ks from its X^k) it was proposed to exclude latent sequence combinations from X^k to Z^k which are not possible according to the given temporal structure of the TSCBN. Any drawn MSSs Z^k is valid if the following is given:

- If a sequence element v_{ik} has an identical state as its previous intra-dependent node ($v_{ik} = v_{i(k-1)}$), the end-time $t_{abs}^{i(k+1)}$ of its interval needs to happen after all its parents' v_{rt} start times t_{abs}^{rt}: $t_{abs}^{i(k+1)} > t_{abs}^{rt}$.
- For all parents v_{rt} with $v_{rt} = v_{r(t-1)}$ of a sequence element v_{ik} with $v_{ik} \neq v_{i(k-1)}$, v_{rt}'s start times need to occur before v_{ik} ends. That is, $t_{abs}^{i(k+1)} > t_{abs}^{rt}$.
- For all parents v_{rt} with $v_{rt} \neq v_{r(t-1)}$ of nodes v_{ik} with $v_{ik} \neq v_{i(k-1)}$, v_{rt} it is required that $t_{abs}^{ik} < t_{abs}^{rt}$.

EM State node estimation: To compute the expectation per TV exactly all combinations of mappings from X_i to Z_i would need to be computed. Thus, efficient exact computation of $\mathbb{E}_{Z \sim p(Z|X,\Theta)}[\log p(X, Z|\Theta)]$ is not possible. To solve this, a Monte Carlo sampling approximation is used to determine approximate distributions at each step.
Using K observations and M sequences (i.e. TVs) EM is performed as follows. Per observation k and sequence i a local estimate $q(Z_i^k)$ of the latent sequence that can be mapped from the corresponding observed sequence X_i^k is assumed (e.g. given X_i^k = ABC and 4 nodes for TV S_i, Z_i^k may be ABBC, AABC or ABCC), where Θ holds the parameters of the model at the current iteration.
At first both $q(Z_i^k)$ and all Θ are assumed uniformly distributed, before being updated on each iteration. For this, per observation MCMC sampling from all M $q(Z_i^k)$s is used to draw n_{samp} samples from the whole TSCBN. This gives n_{samp} (or less, as samples not satisfying temporal constraints are dropped) valid outcomes per local latent sequence Z_i^k. With this the following update steps are performed.

Update $q(Z_i^k)$: To maximize the likelihood of the local estimate, the updated estimate $q^*(Z_i^k)$ is computed as $q^*(Z_i^k) = \sum_{t=1}^{n_{samp}} p(Z_{it}^k|\Theta)$, where Z_{it}^k is the t-th sample drawn (e.g. $\langle A, A, B, C\rangle$) and $p(Z_{it}^k|\Theta)$ is the likelihood of this sequence to occur.

Update Θ: To update Θ, $\mathbb{E}[\log p(X, Z|\Theta)]$ is found by simply counting all occurrences of all n_{samp} drawn samples under the respective conditioning observed in the sample. With this the likelihood is iteratively maximized.

7.3.6.5 Variational Inference

A further approach, that we proposed in [8], is Variational Inference (VI).
There the framework of [247] was extended to introduce a VI approach for TSCBN. VI tries to estimate the posterior latent distribution and the posterior observed distribution. In VI this is achieved by approximating the global latent posterior distribution $p(Z|\Theta)$ with tractable local distribution estimates $q(Z)$. This is done through maximization of the Evidence Lower Bound (ELBO). Further, using the Mean field assumption the local estimates are considered as conditionally independent, which allows to compute individual latent variables separately. In VI the latent nodes v_{ij} and Δt_{ij} are considered together.

Structure and mapping of latency: For the VI approach the probabilistic structure shown in Figure 7.4 is assumed, but with inverted edges reaching from v_{ij} to X_i for all TVs. From this representation, update equations for all RVs v_{ij} and Δt_{ij} where deduced. Notably, in this representation observations are mapped to latent state node values. It thus, is assumed that each observation X_o only allows for a subset of valid latent outcomes, which are represented in each v_{ij} of each X_i. Thus, per observation $X_o = (x_{o1}, x_{o2}, ...)$ only a subset of mappings from v_{ij} to X_i are possible, e.g. for an observed $X_i = \langle A, B \rangle$ Z_i might only be $(v_{i1} = A, v_{i2} = A, v_{i3} = B)$ or $(v_{i1} = A, v_{i2} = B, v_{i3} = B)$. All such valid mappings from an observed $X_i = X_o$ to its latent Z_i were defined as

$$p(X_i = X_o|Z_i) = \begin{cases} 1 \; \mathit{if} \; Z_i \mapsto X_o \; \mathit{valid} \\ 0 \; \mathit{else} \end{cases} \tag{7.27}$$

With this, invalid MSSs combinations (that are formed by all Z_i combined) have a likelihood of 0 in the TSCBN.

ELBO derivation: In VI the goal is to find the variational distribution $q(Z)$ that approximates the true posterior $p(Z|X)$ of the TSCBN.
According to [247] the likelihood of a general latent BN can be written as

$$p(Z, X) = \prod_n p(w_n|Pa(w_n)) \tag{7.28}$$

, with w_n being all n nodes in the network, i.e. in TSCBNs either state nodes v or temporal nodes Δt.
Further, the probability $P(X)$ of any observation can be written in terms of the ELBO $\mathcal{L}$ and the KL divergence between the real and variational distribution as

$$ln(P(X)) = \mathcal{L}(q) + \mathbb{KL}(q||p) \tag{7.29}$$

$$\mathcal{L}(q) = \sum_Z q(Z|X) \cdot ln(\frac{p(Z, X)}{q(Z|X)}) \tag{7.30}$$

$$\mathbb{KL}(q||p) = -\sum_Z q(Z|X) ln(\frac{p(Z|X)}{q(Z|X)}) \tag{7.31}$$

, where $\sum_{Z}$ is the sum over all network outcomes of Z [247]. With this, minimization of $\mathbb{KL}$ is equal to maximizing the ELBO which can be written as expectation

$$\mathcal{L}(q) = \mathbb{E}_{q(Z)}[\log p(X, Z) - \log q(Z)] \tag{7.32}$$

Mean Field Approximation: The variational JPD $q(Z)$ is factorized to allow for local computations around nodes as

$$q(Z) = \prod_{i=0}^{M} \prod_{j=0}^{M_i} q(z_{ij}) \tag{7.33}$$

where z_{ij} can be a discrete state node v_{ij} or a continuous temporal node Δt_{ij}.
With this, maximizing the overall ELBO (i.e. finding good approximations of parameters per node) is similar to maximization of the ELBO of the factorized distributions $q(z_{ij})$.

Coordinate Ascent Variational Inference in TSCBNs In [8] it was proposed to use, Coordinate Ascent Variational Inference (CAVI), for the maximization of the factorized distributions $q(z_{ij})$. This approach uses iterative updates to maximize the ELBO. By inserting the factorization of equation 7.33 in equation 7.29, it is found that the updated variational distribution $q^*(z_{ij})$ is

$$q^*(z_{ij}) \propto \exp\left(\mathbb{E}_{q_{-ij}}[\log p(Z, X)]\right) \tag{7.34}$$

Computation of Expectation: Any parameter of a node in the network (v or Δt) are updated by computing this exponential expectation iteratively until convergence. To compute this expectations efficiently the following assumptions are made:

- $\mathbb{E}$ is both computed over K observations and as stated above for all valid latent mappings from $X_i \mapsto Z_i$. Mathematically this results from equations 7.27, while computationally this means to compute $\mathbb{E}$ over valid combinations only. This reduces computational costs.

- The computational costs are further optimized by enabling local computations per node. This is, as according to [247] the expectation needs to only be computed over the Markov Blanket around the updated node z_{ij}.

Using those assumptions update equations for $q^*(z_{ij}^k)$ of state nodes v_{ij} and for temporal nodes Δt_{ij} were found. This is done by iteratively alternating between updating network parameters $p(Z|X)$ from all $q^*(z_{ij})$ and updating $q^*(z_{ij})$ from the current network parameters $p(Z|X)$. This is done as follows.

Update State Nodes v_{ij}: With above assumptions and the definition of the TSCBN structure the update equation of any state node v_{ij} is given as

$$q^*(v_{ij}) \propto \exp\left(\mathbb{E}_{q_{-ij}}[\log p(Z,X)]\right) \tag{7.35}$$

$$q^*(v_{ij}) \propto \exp\left(\mathbb{E}_{q_{-ij}}[\log\left(\Psi(v_{ij}) \cdot \Gamma(v_{ij}) \cdot \Xi(v_{ij})\right)]\right) \tag{7.36}$$

with Ψ as the factor with all parents of v_{ij}, Γ as all factors that contain co-parents of v_{ij} and Ξ as factor with children of v_{ij}, which is

$$\Psi(v_{ij}) = p(v_{ij}|Pa(v_{ij})) \tag{7.37}$$

$$\Gamma(v_{ij}) = \prod_{\substack{\gamma \in CoPa(v_{ij}) \\ \xi \in Ch(v_{ij})}} p(\xi|\gamma) \tag{7.38}$$

$$\Xi(v_{ij}) = \prod_{\xi \in Ch(v_{ij})} p(\xi|v_{ij} \cup CoPa(\xi)) \tag{7.39}$$

Here, $\Xi(v_{ij})$ includes the observed sequence X_i with factor $p(X_i|v_{ij} \cup CoPa(v_{ij}))$, which was defined in equation 7.27 to be zero for invalid mappings. Thus, invalid mappings can be ignored in this equation when computing the expectation.
Computation of $\mathbb{E}$*:* In this update equation all expectations are composed of terms that are solely discrete and terms that are of mixed type (i.e. continuous and discrete). This is, as all nodes in a TSCBN also have a temporal node that needs to be included in this step. For the purpose of clarity the expectation was split in a mixed $\mathbb{E}_c$ and a discrete part $\mathbb{E}_d$ which gives the typical shape of the update equation

$$\mathbb{E}[\log p(Z,X)] = \mathbb{E}[\log(p(\Delta t_{ij}|v_{ij},...) + \log(p(\Delta t_{kr}|v_{ij},...) + ... \tag{7.40}$$

$$+\log(p(v_{ij}|v_{kr},...)] + \log(p(v_{xy}|v_{qs},...)] \tag{7.41}$$

$$= \mathbb{E}[\log(p(\Delta t_{ij}|v_{ij},...) + \log(p(\Delta t_{kr}|v_{ij},...) + ...] \tag{7.42}$$

$$+\mathbb{E}[\log(p(v_{ij}|v_{kr},...)] + \log(p(v_{xy}|v_{qs},...)] = \mathbb{E}_c + \mathbb{E}_d \tag{7.43}$$

$\mathbb{E}_d$ can be simply computed by iteration of outcome combinations, while $\mathbb{E}_c$ is computed as follows. As all Δt are root nodes in TSCBNs, $\mathbb{E}_c$ is made up of components of the shape $\mathbb{E}[\Delta t|Pa(\Delta t)]$, with discrete nodes $Pa(\Delta t)$. Those components are computed as

$$\mathbb{E}[\Delta t|Pa(\Delta t)] = \sum_V (\int_{-\infty}^{\infty} \Delta t \cdot \log p(\Delta t|Pa(\Delta t)) \mathrm{d} \; \Delta \; \mathrm{t}) \cdot q(Pa(\Delta t)) = \tag{7.44}$$

$$\sum_V \chi(\Delta t, Pa(\Delta t)) \cdot q(Pa(\Delta t)) \tag{7.45}$$

where V are all outcome combinations of the discrete parents of Δt, i.e. $Pa(\Delta t)$, and $q(Pa(\Delta t))$ its local probabilities. Notably, in CAVI the expectations are computed with respect to the update node, i.e. $\mathbb{E}_{q_{-ij}}$ is excluding $q(v_{ij})$.

Updating network parameters of v_{ij}: With the local updated estimates the network parameters can be found with

$$p(v_{ij}|Pa(v_{ij})) \approx mean(\prod_{\tau \in \{v_{ij}\} \cup Pa(v_{ij})} q(\tau)) \tag{7.46}$$

Here, *mean* is the mean over all K observations. This is possible according to the mean field assumption where it is assumed that all nodes $q(v_{ij})$ are pairwise independent.

Update temporal nodes Δt_{ij}: Here, for nodes Δt_{ij} still the structure in Figure 7.4 is considered and a Gaussian distribution per temporal node was assumed. The update equation for the local estimate of Δt_{ij}, governed by its parameters μ and σ per observation are presented.

$$q^*(\Delta t_{ij}; \mu^*_{ij}, {\sigma^*_{ij}}^2) \propto \exp\left(\mathbb{E}_{\Delta t_{-ij}}[\log p(Z, X)]\right) \tag{7.47}$$

For this case again the structure of TSCBNs is assumed and only the Markov Blanket is considered per node, i.e. as Δt_{ij} are leaf nodes only one term with its parents is of relevance. Also, the valid mappings assumption in equation 7.27 holds. That gives the non constant part to compute the expectation over as

$$p(Z, X) = p(\Delta t_{ij}; \mu_{ij}, {\sigma_{ij}}^2 | Pa(\Delta t_{ij}))) \tag{7.48}$$

, where the remaining term captures the distribution given its parents. This yields

$$q^*(\Delta t_{ij}; \mu^*_{ij}, {\sigma^*_{ij}}^2) \propto \exp\left(\mathbb{E}_{\Delta t_{-ij}}[\log p(\Delta t_{ij}; \mu_{ij}, {\sigma_{ij}}^2 | Pa(\Delta t_{ij})))]\right) \tag{7.49}$$

Estimating latent outcomes from observations: It is assumed that the variances ${\sigma_{ij}}^2$ are fixed and only update each μ_{ij} per outcome combination ω that $Pa(\Delta t_{ij})$ can take. Also, for the computation of the expectation per outcome combination and observation the absolute times of observed state changes to latent observed values are mapped for each node Δt_{ij} by using the method described above. This allows to get a full sample estimate of the network (with values for all Δt) for each outcome combination ω which is based on the current observation. With this latency in mappings from observed vectors X_i to its latent node outcomes is handled.
Per such estimated observation $\Delta t_{ij\text{obs}|\omega}$ a Gaussian $\mathcal{N}(\mu_{ij|\omega} = \Delta t_{ij\text{obs}|\omega}, \sigma^2)$ is assumed. The idea is to compute the expectation around the observed absolute times by interpolating them and by resolving latency via computation of the CAVI update around relevant parent outcomes.

Estimation of local node estimate $q^*(\Delta t_{ij}; \mu^*, \sigma^{*2})$: From the interpolated observations $\mu_{ij|\omega}$ and the current estimates of the outcomes $q(v_r = \omega_r)$ a local estimate per node is found with equation 7.47. For this the expectation can be written as

$$\mathbb{E}_{\Delta t_{-ij}}[\log p(Z,X)] \tag{7.50}$$

$$= \sum_{\omega \in Pa(\Delta t_{ij})} \log p(\Delta t_{ij}; \mu_{ij}, {\sigma_{ij}}^2 | Pa(\Delta t_{ij}) = \omega) \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r) \tag{7.51}$$

$$= \sum_{\omega \in Pa(\Delta t_{ij})} \log \mathcal{N}(\mu_{ij|\omega}, \sigma^2) \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r) \tag{7.52}$$

, where v_r are all nodes of $Pa(\Delta t_{ij})$ with a specific outcome ω_r, e.g. for a node Δt_{12} v_r might correspond to nodes v_{12} and v_{11}. Also the Gaussian distribution is fixed with a mean of the given interpolated observation $\mu_{ij|\omega} = \Delta t_{ijobs|\omega}$.
When, inserting equation B (provided in the appendix) into equation 7.47 it is found that for the local estimate μ^*_{ij} of the temporal node Δt_{ij} it holds that

$$\mu^*_{ij} = \frac{\sum\limits_{\omega \in Pa(\Delta t_{ij})} \mu_{ij|\omega} \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)}{\sum\limits_{\omega \in Pa(\Delta t_{ij})} \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)} \tag{7.53}$$

This makes intuitively sense, as the μ_{ij} governing Δt_{ij} is a weighted average over its outcomes depending on its likelihoods. A full derivation of this update can be found in Appendix B.

Updating network parameters of Δt: Finally the estimate $p(\Delta t_{ij}; \mu_{ij|\omega}, \sigma^2_{ij} | Pa(\Delta t_{ij})$ of each CPD of each temporal node can be found as the weighted average

$$\mu^*_{ij|\omega} = \frac{\sum\limits_{\omega \in Pa(\Delta t_{ij})} \mu^*_{ij} \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)}{\sum\limits_{\omega \in Pa(\Delta t_{ij})} \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)} \tag{7.54}$$

Estimation Algorithm: With the given update equations the VI algorithm in Alg. 7.3.6.6 was deduced. Notably, the observations X are included in the way described in this section at lines 3 and 6. ω resembles all possible parent outcome combinations.

7.3.6.6 Computing Δt

When computing any Δt_{ij} in a latent sequence (e.g. $\langle A, A, B, B\rangle$) from a full observation X_i (e.g. $\langle A, B\rangle$), in order to be usable in the computation of the expectations in CAVI each latent node requires a defined absolute time at which it occurs. The following interpolation strategy is used for this.

- Times between two elements are interpolated linearly within known parts, e.g. given A at $t = 1$ and B at $t = 2$ would result in times and values $\langle (A, 1), (A, 1.33), (A, 1.66), (B, 2.0)\rangle$ for a latent sequence $\langle A, A, A, B\rangle$.

Algorithm 2 CAVI Approach for parameter estimation in TSCBNs [8]

```
Input: $V$: State Nodes, $T$: Temporal nodes, $X$: observations
Output: $\mu^*_{ij|\omega}$: Estimated parameters of temporal nodes, $p^*(v_{ij}|Pa(v_{ij}))$ : Estimated Parameters of state nodes
1: $q(v_{ij}|\omega) = \mathcal{U}(); \forall v_{ij} \in V$    ▷ Initialization
2: while $\neg$ ($\mathcal{L}(q)$ converged) do
3:     $q^*(v_{ij}) = \exp(\mathbb{E}_{q_{-ij}}[\log(\Psi(v_{ij}) \cdot \Gamma(v_{ij}) \cdot \Xi(v_{ij}))]); \forall v_{ij} \in V$    ▷ State Node Updates
4:     $p^*(v_{ij}|Pa(v_{ij})) = mean(\prod_{\tau \in \{v_{ij}\} \cup Pa(v_{ij})} q^*(\tau)); \forall v_{ij} \in V$    ▷ State Network Updates
5:
6:     $\mu^*_{ij} = \frac{\sum_{\omega \in Pa(\Delta t_{ij})} \mu_{ij|\omega} \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)}{\sum_{\omega \in Pa(\Delta t_{ij})} \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)}; \forall \Delta t_{ij} \in T$    ▷ Temporal Node Updates
7:     $\mu^*_{ij|\omega} = \frac{\sum_{\omega \in Pa(\Delta t_{ij})} \mu^*_{ij} \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)}{\sum_{\omega \in Pa(\Delta t_{ij})} \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)}; \forall \Delta t_{ij} \in T$    ▷ Temporal Network Updates
8: end while
9: return $\mu^*_{ij|\omega} \forall \Delta t_{ij} \in T$, $p^*(v_{ij}|Pa(v_{ij})); \forall v_{ij} \in V$
```

- If the last observed element is followed by further elements in the latent sequence, its preceding element's distance is used for interpolation, e.g. having seen A at $t = 1$ and B at $t = 2$ that is extended to $\langle A, A, B, B, B\rangle$ would result in $\langle (A, 1), (A, 1.5), (B, 2), (B, 2.5), (B, 3.0)\rangle$. This extension time is determined by subtracting the last observed time by its predecessors interpolated time. If no predecessor is available the distance is computed by dividing the total time of the current sequence by its number of nodes. While this interpolation may potentially lead to temporal inconsistency, it produces a reasonable approximation that tends to decrease deviation from the true mean when enough sequences were observed.

7.4 Evaluation

Lastly, a comparison of models, parameter estimation methods and structure discovery approaches was presented. In this work the results of this evaluation are revised to show the suitability of TSCBNs to be used for representation of MSSs in the proposed DM pipeline. Further, the comparison of parameter estimation and structure discovery approaches is revised for the same purpose. The following is thus, taken from [8].

7.4.1 Experimental Setup

All experiments were conducted on an HP™ Z-840 equipped with Intel® Xeon® E5-2640 v3 2.60GHz CPUs and 96 GB of RAM. To compare the models, a synthetic evaluation was performed, where ground truth MSSs are generated and then, are used for training and evaluating the models.

Data Generator: The following data generator was used to generate MSSs for this purpose. This generator is a TSCBN, with defined structure that models evolving dependencies between TVs. Sampling from such a defined TSCBN gives a set of MSSs that is used as input for training of all models. Also, for the comparison of the models the generator TSCBN is used as a ground-truth. The state lengths are sampled from a Gaussian distribution. Additionally, the generator allows to set a defined probability of state change per node.
From this model defined numbers of samples were drawn which are then, used for evaluation. Those samples are latent sequences per TV (e.g. $\langle S_0 = 0, S_0 = 0, S_0 = 1\rangle$) when drawn, which are reduced to true observations (e.g. $\langle S_0 = 0, S_0 = 1\rangle$).
The parameters that define the MSSs are set by defining the TSCBN structure. This includes the following parameters.

- **Structure of TSCBN:** number of TVs n_{TV}, number of nodes per TV n_n, i.e. length of sequence per TV
- **Connections in TSCBN:** number of TVs that have a connection with other TVs n_{TVint}, number of edges n_{inter} that nodes have if a connection is present.
- **Parameterization of TSCBN:** number of states per state node n_c and settings for μ and σ per temporal node, state change probability p_{SC}, which is the probability with which a node remains in the state it previously was in.

Further, to allow for comparison, the structure of DBNs and CTBNs is deduced from the true structure of the TSCBN. With this the following experiment were performed.

1. A TSCBN with n_{TV} TVs and n_n nodes per TV is created. A DBN with n_{TV} nodes per time slice and a CTBN with a fixed structure with n_{TV} nodes are generated.
2. Per TV, n_{TVint} connecting TVs are randomly chosen. Then, n_{inter} connections to nodes of this TV are randomly set. In CTBNs and DBNs those edges form the static network structure. This structure is assumed given for DBNs, CTBNs and TSCBNs when performing parameter estimation. In DBNs further the structure is defined by its resolution, which is the breadth of each time slice.
3. Next, n_c states are created per node and a CPD is randomly generated per node and condition. In TSCBNs, this CPD forms the ground truth and is used for sampling MSSs. Each CPD is created such that all probabilities of a intra-node to remain in its same state are set fixed to p_{SC}, while a random distribution is drawn for the remaining CPD entries. Further, a Gaussian distribution with μ' and σ' is used to draw values μ, that are set to the temporal nodes under given conditions. With this, per node different μs are provided and state sequences of equal lengths are avoided. In the experiments σ'^2 was set to 0.1.

With this, a structure for all three network types is given, while for TSCBNs an additional parameterization is given. The TSCBN is used for sampling sequences as described in the beginning of this section. Next, a copy of the TSCBN is made, where parameters are deleted. This TSCBN is used for parameter estimation, while the original is kept as ground truth.

The generated samples are split 90:10 into a training and a test set which are randomly chosen from the data. Then, for all three network types parameter estimation is performed by using the sampled MSSs from the training data set.
For the implementation of DBNs [192] the implementation of python's libpgm package was used and for the implementation of CTBNs [193] a python wrapper for the R package CTBN-RLE [248] was written and used. In DBNs static edges between related TVs are defined and the the structure is repeated in discrete time, with outcomes at each slice. The maximum distance of a time-slice to a state change to capture is defined as *DBN tolerance*, which defines the resolution of the structure repetition. Parameter Estimation is done with a Maximum Likelihood Estimator. For CTBNs the same static edges between TVs are used for both the transition and intensity matrices as provided by the ground truth TSCBN. Then, CTBN-RLE's parameter learning engine is used to estimate parameters of the CTBN.

Parameters used: For all models parameters are 5 nodes per TV (i.e. 4 intervals), 4 states per node, per TV 2 edges to two other TVs. For the DBN a resolution of 0.02 was used and the length of all intervals is drawn from a Gaussian with $\mu = 0.5$ and $\sigma^2 = 0.1$. The number of TVs n_{TV}, the number of samples for training and test n_{samp}, and the probability of a state change occurring p_{SC} were varied. For parameter estimation per sequence during MCMC sampling 1000 samples are drawn per iteration, 5 iterations were performed for EM and VI. Per iteration a CPD smoothing of $\epsilon = 0.1$ is used.

7.4.2 Model

With the above experiments the structure of TSCBNs was evaluated. For the proposed DM pipeline TSCBNs are used as it provides a compact yet expressive representation to represent MSSs under uncertainty. This model requires a little more parameters than CTBNs, but is higher in expressiveness. This was shown in [8] and relevant experiments for comparison of structures for the three models are discussed in this context in the following.

7.4.2.1 Setup

Evaluation Criteria: The model structure is compared in terms of the number of edges n_E, nodes n_N, states n_S and parameters n_C. The structure of the models is quantified with the number of components required to model a MSS. For a given model this includes the number of edges n_E, number of nodes n_N, total number of states n_S and the total number of entries required in all CPD tables of all nodes n_C (i.e. the number of parameters).

Experiments: To evaluate the model *structure*, the number of intervals and TVs is varied and an according TSCBN and DBN found to represent this MSS. The structures are evaluated in terms of number of nodes, edges, states and conditional probability entries (=parameters).

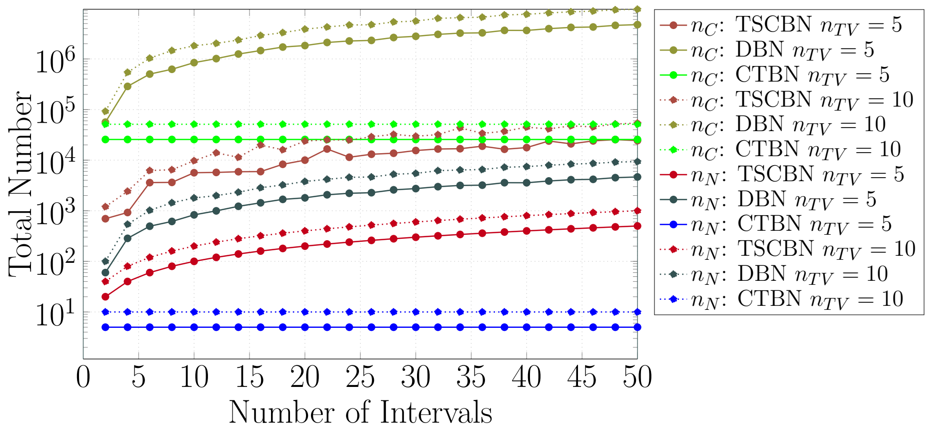


Figure 7.12: Results of structural complexity, as number of nodes n_N and CPDs n_C, for various numbers of TVs [8].

7.4.2.2 Results

Figure 7.12 shows the results of the experiments. It can be seen that CTBNs and TSCBNs are both light in terms of parameters required for representation. Thus, in terms of parameter complexity both equally suited to represent MSSs. In contrast to that, DBNs quickly explode in complexity and are thus, less suited to be used in the proposed DM pipeline.
This comparison of structural complexity was explained per model.
DBNs require additional nodes n_N per interval. Also, DBNs add multiple nodes per state change (depending on its resolution). With growing precision this results in increase of parameters. CTBNs have a fixed number of nodes, i.e. only the TVs and its dependencies. In terms of parameters n_C intensity matrices are stored per condition, which increases in complexity with more parents per TV. In TSCBNs per additional interval an additional node is required. However, in contrast to DBNs only state changes are modeled, making it compact in representation.
Thus, CTBNs are similarly light as TSCBNs in terms of parameter size, if no evolving dependencies are given, while TSCBNs show to be lighter when evolving dependencies are modeled in MSSs. Especially, when assuming evolving dependencies, TSCBN require edges only at nodes within the process where actual dependence is given. With this less edges and states per node are required to model MSSs, as less conditional combinations are possible per node. In contrast to that both, in DBNs and CTBNs all dependencies between TVs need to be directly given, which results in state explosions in those cases. For the case of Specification Mining it is required to model functional procedures, which are of a defined structure with changing dependencies. This evaluation shows that TSCBNs are well suited for those tasks as the number of TVs, parameters and nodes is low. As CTBNs are similarly light weight those are equally well suited. However, due to its lack of expressiveness for the proposed DM pipeline TSCBNs are preferred.

7.4.3 Structure Discovery

Multiple structure discovery approaches can be used for discovery of TSCBNs. Some prominent representatives where compared against TrieDiscover. It was shown that TrieDiscover is best suited among those approaches for learning TSCBNs from MSSs. Results of [8] that justify this are discussed here.

7.4.3.1 Setup

The described generator is used for verification with the generated models as ground truth.

Compared Approaches: Six approaches were compared.

- **TrieDiscover:** All three variants cbTD, cbvTD and sbTD are used.
- **Greedy Hill Climbing (GHC):** An optimization-based approach.
- **PC Algorithm:** A classical constraint-based approach.
- **Max-Min Hill Climbing Algorithm (MMHC):** A hybrid max-min hill climbing algorithm.

To make GHC, PC and MMHC applicable to learning of TSCBNs the assignment of state changes to temporal nodes in the TSCBN has to be done prior to execution. For this a naive assignment (i-th event of signal X is assigned to node X_i) is used and it was shown that the advanced event assignment of TrieDiscover gave indeed better results in expressiveness. Also, edges in non-TD approaches are oriented using the index number of the sequential nodes per TV (from smaller index number to the larger one).

Parameterization: The following parameters were used. The temporal gap between subsequent events of a signal is randomly drawn between 0.5 and 1.0 from a uniform distribution. The state change probability is set to 0.95, the number of states per TV to 3, number of TVs to $n_{TV} = 5$ and length per TV to $n_L = 4$ and $0.5 \cdot n_L \cdot n_{TV}$ random inter-edges and 5000 MSSs sampled for learning. The structure learned by each algorithm was compared to the structure of the original network. TrieDiscover is parameterized with $k = 0.1$ to be able to filter the noisy parts, t_{th} to 1.0 as gaps between intra nodes are drawn from Gaussians with mean 0.5. In the sbTD BIC is used as score, in cbTD and cbvTD $\alpha = 0.01$ and $\chi_{thr} = 1.0$. The significance level in the constraint-based and hybrid approaches was chosen from the values $\alpha = 0.01$, $\alpha = 0.05$, and $\alpha = 0.1$. Good results were achieved when using $\alpha = 0.01$ in the PC, MMHC, cbTD and cbvTD algorithms. Also, a maximal condition set size of two nodes turned out to be a good trade off between precision and performance when executing the cbTD and PC algorithm. All algorithms were implemented in Python, while GHC, PC, and MMHC were taken from the Python package pgmpy[1].

[1]`http://pgmpy.org/`

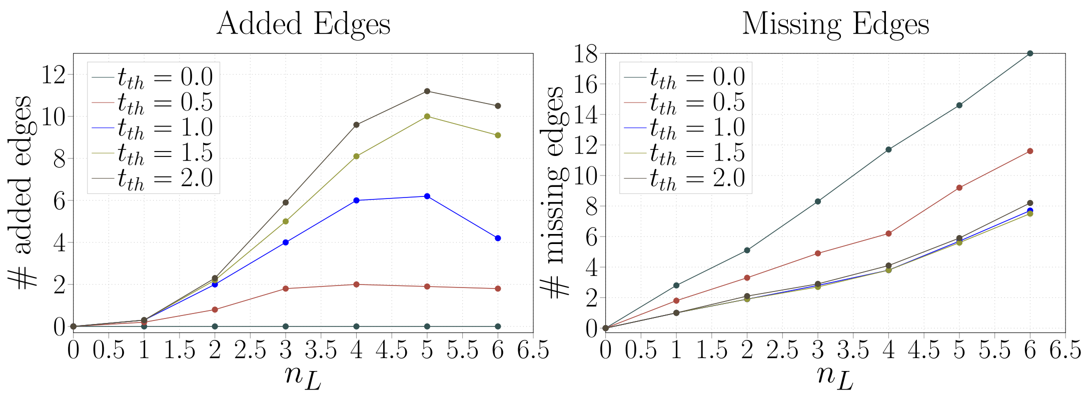


Figure 7.13: Left: Various number of added and missing edges for sbTD in comparison to ground truth, when structure and t_{th} is varied, with 3 TVs assumed [8].

Evaluation Criteria: For the evaluation the following criteria were used.

- **Run time:** To measure computational efficiency the run time is measured.
- **Additional Edges:** This is the number of edges that are present in the discovered structure, but cannot be found in the original network.
- **Missing Edges:** This is the number of edges that are missing in the discovered structure, but are found in the original network.
- **Structural Hamming Distance (SHD):** Introduced in [233], this metric is used to compare the discovered to the original network structure. It is given by the sum of the number of missing edges, the number of additional edges and the number of wrongly oriented edges. Thus, a small SHD indicates a higher similarity between networks.

Experiments: The following experiments were performed. At first parameterization of TrieDiscover was inspected. That is, in the first experiments in Figure 7.13 the influence of t_{th} on the structure and the run times of the individual steps shown in Figure 7.14. All approaches were compared against each other for different numbers of samples and different SC probabilities, with results shown in Figure 7.15 and 7.16.

7.4.3.2 Results

Varying Parameters of TrieDiscover: As Figure 7.13 shows t_{th} is a main factor for the number of edges in the learned structure. This is, as the parent candidate set for final structure optimization is dependent on it. It can be seen that small values (e.g. $t_{th} = 0.0$) lead to less edges (many missing edges), as this set is empty, while a higher t_{th} reduces the number of missing edges. Also, a bigger t_{th} leads to an increase of added edges, as the structure optimization step yields a large number of edges that are spurious dependencies, e.g. given an edge $A_2 \rightarrow B_2$, strong dependencies exist between A_2 and preceding events. This may lead to spurious dependence between A_1 and B_2. A good

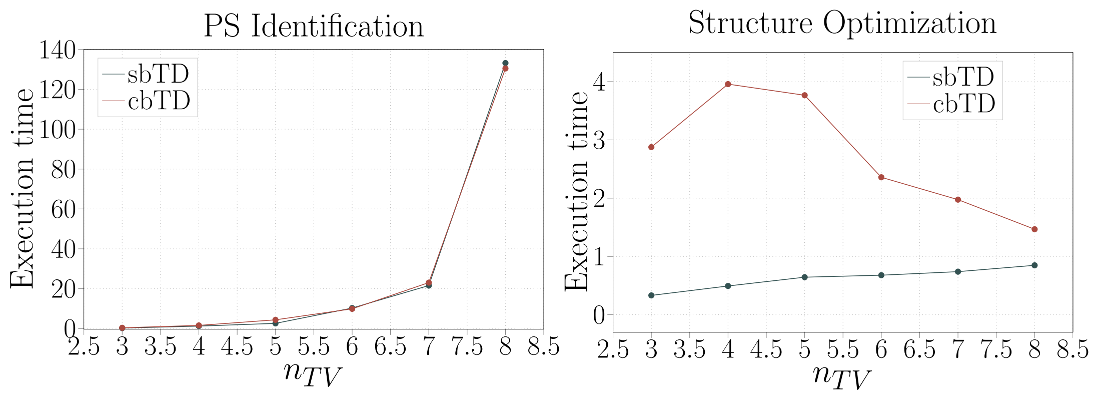


Figure 7.14: Execution time when assuming $n_L = 3$ for various numbers of TVs and the two steps of TrieDiscover [8].

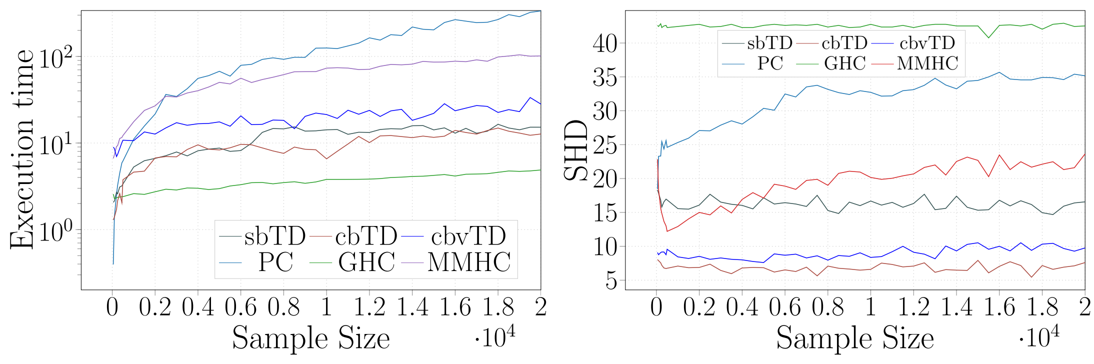


Figure 7.15: Results in terms of run time and SHD for various sizes of the training set, SC probabilities [8].

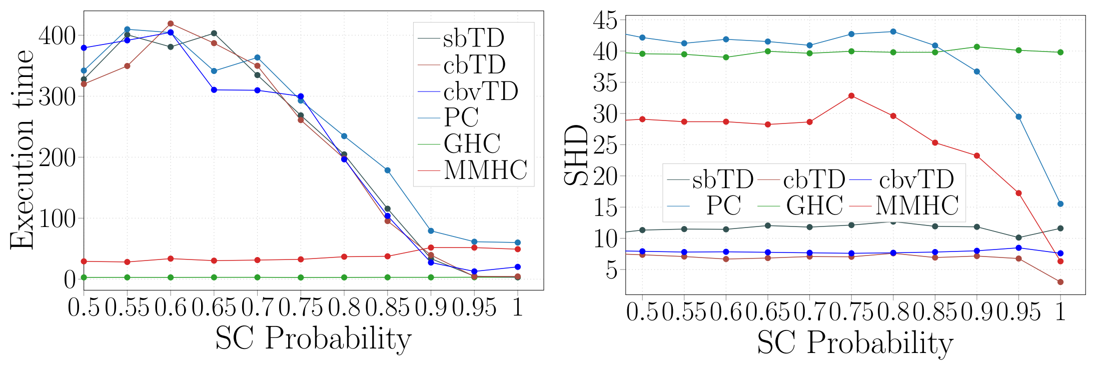


Figure 7.16: Results in terms of run time and SHD for various sizes of the training set, SC probabilities [8].

way to find appropriate values for t_{th} is by empirically evaluating gaps between state changes, which will be shown in Chapter 9.

Varying Number of TVs: As Figure 7.14 illustrates, the run time increases with more TVs that are considered. However, in the given experiments no parallelization was used for PS identification, while many steps such as merging are done in parallel which allows to scale TrieDiscover. For instance, assuming that 7 TVs require 20 seconds when using one core, then, in theory 40 CPUs might handle 280 TVs within a similar time range. However, this is yet to be shown in future work. Still for the prototype of the DM pipeline the given sequential implementation is sufficient. In general the execution time is mainly influenced by the number of TVs, length of the sequences and the choice of the parameters. In particular, parent set identification becomes increasingly complex and makes up around 95 % of run time. The shown run time assumes all TVs to have 4 nodes. In real world procedures however, the interval number varies across TVs. Thus, it is assumed, that more TVs with various interval numbers are handled in similar run time.
Nevertheless, as can be seen in this evaluation reducing the number of TVs as it is done here in the TV clustering step of the proposed DM pipeline is an essential step to provide reasonable execution times. In practice finding specifications among 10 TVs is already meaningful, while with parallelization and improvement of this step more TVs could be handled in the future.

Varying Sample Size: The size of the training set does not meaningfully improve precision as Figure 7.15 shows. This is due to the fact that TrieDiscover, is lossless and less samples suffice to discover the according structure. By including expert input for cases of too less samples for correlation detection meaningful structures are found for such critical edges.

Varying SC Probability: The SC probability sets the noise in the data during learning. As Figure 7.16 shows the SHD curve shows that TrieDiscover performs worse for smaller SC probabilities, i.e. if more latency is present. However, overall all three variants of TrieDiscover yield the best results here. Especially if a higher degree of latency is present, TrieDiscover still yields good results. This is as the trie is built up exactly, i.e. even if paths are seen rarely, those are present as candidates for structure optimization.

Comparing approaches: Among the compared approaches as can be seen in Figure 7.15 and 7.16, Trie Discover performs best. MMHC and PC improve with more samples. In terms of run time the GHC and MMHC perform best, while all other approaches yield higher run times with more latency. This is as more latency also means more variation in possible paths represented in the data set. Further, TrieDiscover is about 10 times faster than the constraint-based baselines and also good in precision even for less samples given.
In terms of handling noise, TrieDiscover yield the best results here as it is using a compression with an advanced event to node assignment, as described above. The naive event-to node assignment used in the baselines results in unclear relations between two events, e.g. for a sequence $\langle A, A, B, A\rangle$, the second A is latent thus, the third A will be

assumed A_2. Thus, for the case of no latency the static approaches improve significantly performing similar to TrieDiscover.
Further, unlike the Baseline approaches TrieDiscover does not suffer from wrongly oriented edges as it includes temporal information. Thus, in the general case where latency is given it is preferable to use the TrieDiscover approach for the structure learning of TSCBNs within the proposed DM pipeline.
Among the variants of TrieDiscover constraint-based approaches outperform score-based approaches in terms of additional edges as the latter tend to contain spurious dependencies. The further, use CI tests that are performed conditioned on a set of other nodes, which allows to find and remove edges with spurious dependencies. This is also why cbTD works better than sbTD and should be preferred in the DM pipeline.

7.4.4 Parameter Estimation

Lastly, multiple parameter estimation approaches for TSCBNs were introduced. It was shown that all three estimation approaches perform similar in expressiveness, while run time of VI becomes significantly worse for higher structural complexity. Also, the CTBNs, DBNs and TSCBNs were compared in expressiveness and it can be seen that TSCBNs are best suited for representation of MSSs. In this section the experiments of [8] are revised to underline this.

7.4.4.1 Setup

Evaluation Criteria: Expressiveness was measured using the mean log-likelihood and temporal mean log-likelihood, while the performance was evaluated using the run time. Those were defined as follows.

- **Mean Log-Likelihood:** The mean log-likelihood resembles the state expressiveness of a model computed from N given outcomes with M nodes per outcome. Values closer to zero indicate better expressiveness. It is normalized by the total number of nodes M in the model to allow for fair comparison between TSCBNs, DBNs and CTBN. It is defined as

$$l_{mean}(\Theta|X) = \frac{1}{N \cdot M} \sum_{i=0}^{N} \sum_{j=0}^{M} \log P(x_i^j|Pa(x_i^j)) \tag{7.55}$$

  where x_i^j is the j-th outcome and the i-th node of the model. This metric is computed for all discrete nodes x_i^j (i.e. in TSCBNs all nodes v and in DBNs all nodes). The magnitude of this metric is called the Absolute Mean Log-likelihood. In CTBNs this metric is computed by walking through each sample in a sequence and by storing all previous values of the outcomes of TVs as conditions. Then, per sample the transition matrix per TV is used to find the log-likelihood of the TV transitioning into the observed state under the given previous condition. All computed log-likelihoods are then summed up and normalized by the number of summands yielding the Mean Log-Likelihood. This metric resembles how likely a CTBN would have produced the observed sequence.

- **Temporal Mean Log-Likelihood:** If x_i^j is a temporal node ($x_i^j = \Delta t_i^j$), $P(x_i^j|Pa(x_i^j))$ in $l_{mean}(\Theta|X)$ is the Gaussian of the according temporal node in the TSCBN. It measures the temporal expressiveness of a TSCBN. This criterion is called the temporal log-likelihood in the following.
- **Run time:** The execution time required for the parameter estimation is measured.

Experiments: With this the following experiments were performed. The parameter estimation is evaluated by training a model with 90 percent of the data and testing it with 10 percent. During testing the mean log-likelihood of the test sequences are used for evaluation of the trained model. Further, for the TSCBN the temporal log-likelihood and run time of the parameter estimation is measured. This is done once with fixed $p_{SC} = 0.8$ and increasing n_{samp} and once with fixed $n_{samp} = 2000$ and increasing p_{SC}.

7.4.4.2 Results

Here first the performance of TSCBNs and its parameter estimation approaches are discussed and then, the models are compared.

Varying Number of State Changes: As depicted in Figure 7.17 an increasing number of state changes improves the expressiveness of TSCBNs, as less latency is present. With this, less marginalization during estimation is required making it more precise.
For less TVs and more expected state-changes that occur run time of all estimation approaches improves as shown in the right plot. That is as, less latency in EM and MLE-R requires less MCMC sampling and in VI less computation of expectations.

Varying Number of Samples: In Figure 7.18 it can be seen that in TSCBNs more samples improve precision, which is as more samples contain more observed behavior. With this it allows to better handle data fragmentation and reduce the degree of approximation required.
Further, the same effect is seen when estimating the CPDs of temporal nodes of TSCBNs. For a small number of samples the temporal log-likelihood does improve significantly if more samples are added, while it stagnates later. Also, that less TVs yield more expressive results, as in this case it is more likely that unknown samples are similar to observed samples in training.

Comparing model expressiveness: As Figure 7.17 shows DBNs and CTBNs are worse in expressiveness than TSCBNs for modeling MSSs, which does not change for more samples and when changing the probability of state changes. In DBNs this is as the structure is independent of state changes as learning is done across discretized time-slices. CTBNs generalize over state changes by learning generalized intensity matrices that are independent of the change in dependency of the procedure. TSCBNs, model changes only at state changes where those actually occur making this model more expressive. Even for a SC probability of 0.55 TSCBNs are still about 2 times more expressive than CTBNs and about 8 times more expressive than DBNs. Thus, TSCBNs are preferable to express functional procedures with evolving dependencies based on MSSs.

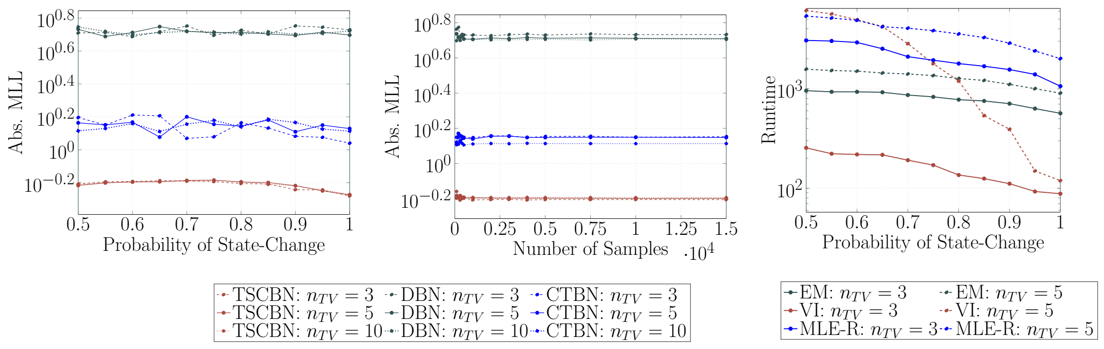


Figure 7.17: *Left and Mid*: Abs. MLL and run time when varying probability of state change and number of training samples using the EM algorithm for estimation of TSCBN parameters. *right:* Run times for three approaches including EM, VI and MLE random for two structure sizes [8].

Comparing Estimation Approaches: Further a comparison of estimation approaches was given, which showed that VI performs best in expressiveness in both state and time, while in run time it is best for a small degree of latency. For increased latency the other approaches perform best. This was shown as follows.
As Figure 7.17 (right) shows, for less latency VI requires less time as in this case computation of expectations is efficiently possible, while MCMC sampling is required in the other approaches. With increased structure and latency more combinations need to be iterated in VI, leading to longer run times. However, in EM and MLE-R bigger structures only increase the time required for sampling and computation of global estimates, which does increase less than the effect of growth in combinations in VI.
Expressiveness in state of all approaches is similar as Figure 7.18 (left) shows. MLE-R seems to perform slightly better than EM and VI. Further, all approaches improve with less latency, as less approximations are required in that case. Also, as Figure 7.18 (mid) depicts with more samples more information is given and thus, improved expressiveness results are found for all approaches.
As VI includes temporal nodes in its computation the temporal expressiveness is best in VI as Figure 7.18 (right) shows. For EM and MLE-R the fitting of a Gaussian curve using the observed samples per condition seems to contain a higher degree of approximation, yielding less expressiveness in time.

7.5 Summary and Conclusion

To be able to extract Specifications and dominant behavior from a set of MSSs an expressive, compact and interpretable model is required. For this three models including DBNs, CTBNs and TSCBNs were compared and described in this chapter. TSCBNs outperform its competitors for this task and thus, show to be best suited for this scenario as a compact structure is given that models functional procedures in terms of state changes at their state and time of occurrence only.
To be able to learn such networks structure discovery and parameter estimation approaches were introduced and compared. For structure discovery three variants of

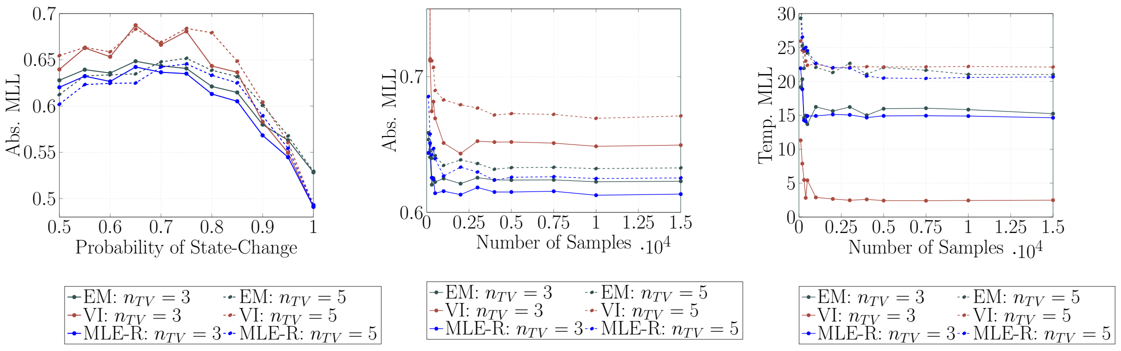


Figure 7.18: The KL divergence and temp. MLL when varying probability of state change and number of training samples for three approaches including EM, VI and MLE random for two structure sizes are shown [8].

TrieDiscover were introduced and compared against MMHC, GHC and PC, showing that TrieDiscover is suited best for this task. In particular its constraint based versions yield better results than the score based ones. For parameter estimation EM, VI and MLE-R were presented. In this chapter the result were revised to show, that all three approaches are similarly well suited to capture the behavior of a set of MSSs. In terms of run time however, VI is preferable for small structures with less latency, while the other approaches are well applicable for bigger structures and more latency.
Consequently, for the penultimate step of the proposed DM pipeline TSCBNs learned with cbTD and EM or VI are preferable. It is well suited to interplay with the other components within this DM pipeline as will be shown in Chapter 9.

8 Inference: Specification Mining and Dominant States

With the steps performed so far complexity was broken down, as a subset of relevant TVs was found and within those subsets functional procedures were identified. Per procedure the distribution of functional procedures is represented in state and time using TCSBNs. Those models represent the observed behavior of a subset of MSSs in a compact and aggregated manner. TSCBNs do this by effectively capturing MSSs. Notably, per group of functional procedures one TSCBN is learned.
Based on those representations it is now possible to extract both dominant behavior and specifications W using the trained model Q which is a TSCBN and an inference I with hyper parameters P_I.

$$W = inf(Q, I, P_I) \tag{8.1}$$

The extracted models represent functional procedures, where irrelevant information such as noise, outliers or TVs were removed in previous steps. Thus, the resulting models contain a clean essence of the observed system behavior of a trace that is of relevance to the expert. As the last step of the proposed pipeline, the main goal of this chapter is to extract dominant behavior and to extract specifications from those models. For this the following two methods are used.

- **BaySpec on TSCBNs:** This approach, that we first introduced in [10], is able to extract a list of LTL specifications from TSCBNs. This list resembles a set of specifications that allow to perform Model Checking on further incoming traces $\mathbf{K}_s$.
- **MPE on TSCBNs:** The Most Probable Explanation (MPE) allows to identify the most likely MSS state that was captured by a TSCBN. That is, the MSSs that were observed most in the functional procedure are most likely to be produced by the network. This allows to identify dominating system states which helps to improve system and data understanding of an expert.

In TSCBNs the model capacity is often higher than can be covered by the number of observed samples. However, as the aim is the extraction of specifications and dominant states in that case, it is assumed that unobserved state constellations of the TSCBNs are less likely than the once that are observed. Thus, by considering most likely behavior in the TSCBNs the inference approach mainly operates on information gained from observations.
In BNs generally the *Markovian Assumption* holds and so it does in TSCBNs. That is, each RV is conditionally independent of its non-descendants given its parents. Thus, in theory no temporal chains of events longer than three RVs should be possible to be captured. However, in the given case it is assumed that similar mostly complete MSSs are

used for training (as it is trained within a defined functional procedure) and thus, overall likely state constellations correspond to variants that were observed. Consequently, paths and likely constellations within the TSCBN can be seen as capturing full observed chains of events.

In order to extract knowledge from such models MPE or BaySpec can be used. These approaches allow to break down a complex high dimensional trace into a set of dominant system behaviors and to semi-automatically produce specifications that are of relevance to the expert. Both approaches are presented in the further scope of this chapter.
Existing approaches for extraction of dominant behavior and for Specification Mining were presented in Chapter 3. As discussed there, those approaches do not allow for multi-functionality, do not find specifications of arbitrary length, are less robust to noise, do not include structural information by not representing MSSs but rather chains of events and do not allow to extract snapshots of dominant behavior of MSSs. This is solved with the proposed approaches.

Chapter Outline: First, preliminaries are discussed in Section 8.1. Next, methods for extraction of dominant behavior are presented in Section 8.2 and in Section 8.3 BaySpec as an approach for Mining of Temporal Specifications from TSCBNs is presented. Lastly, the three approaches are evaluated in Section 8.4. Notably, this chapter is based on the paper that we presented in [10].

8.1 Preliminaries

First, the aim is to identify the most probable behavior that was captured in the TSCBN, which is done with existing approaches that are presented here. Second, in order to understand the proposed Specification Mining procedure a short overview on Model Checking is given.

8.1.1 Inference in Bayesian Networks

It is crucial in Bayesian reasoning to identify and communicate relevant state constellations of a learned BN to allow for meaningful conclusions. A major field that allows to do this both with and without evidence is Bayesian inference of the most probable explanation of a set of hypothesis [22]. There, the task is to identify a most probable joint value assignment of RVs to its values [22].

Explanation: The term explanation depends on the task to be performed and is categorized according to content (what is to be explained), communication (how is the result communicated) and adaption (To whom is the explanation addressed). In the case of TSCBNs the explanation refers to the dominating system state of a set of MSSs, which is the most likely MSS, which is the constellation of the TSCBN with highest likelihood. The process of obtaining this type of explanation is called abduction [249, 250]. Thus, in the given case content is the dominating system state, communication is done by outputting a list of k most probable states and the adaption refers to a domain-specific expert who is familiar with the underlying system behavior that is represented in the

TSCBN.

MPE-problem: Finding the most probable explanation for a set of RVs in a BN is in general an NP-hard problem and was studied in literature under different names (e.g. Most Probable Explanation [251], Maximum Probability Assignment [252], Belief Revision [253], Scenario-Based Explanation [254], Abductive Inference, Maximum A Posteriori Hypothesis [255]) and variants (e.g. with full or partial evidence) [22]. In more recent works, if all RVs are included the most common term is MPE and for partial evidence the term Partial MAP. In this work the focus is on finding the MPE for the given TSCBN. While finding an optimal estimate for this is intractable, the aim is to find a locally optimal solution for this. According to [22], the formal definition of the term MPE is the following.

MPE: Assumes a probabilistic network $B = (G, \Theta)$, where RVs $\mathbf{V}$ are partitioned into a set of evidence nodes $\mathbf{E}$ with a joint value assignment $\mathbf{e}$, and an explanation set $\mathbf{M}$. The output of this problem is the most probable joint value assignment $\mathbf{m}$ to the nodes in $\mathbf{M}$ and evidence $\mathbf{e}$, or $\perp$ if $Pr(m, e) = 0$ for every joint value assignment $\mathbf{m}$ to $\mathbf{M}$. That is, it seeks to find

$$\arg\max_{\mathbf{m}} Pr(\mathbf{m}, \mathbf{e}) \tag{8.2}$$

Here, explanation set refers to the set of variables to be explained.

Basic Approaches: A multitude of methods were presented for this task in previous research, which include the following. The first work was presented in [253], where it was assumed that for each value x of a RV X a best explanation exists for all other RVs. Given this information the MPE is identified for x of X by finding the best explanation for the remaining TVs and then choosing the best value of X. This however, only allowed to find two MPEs [256], which is why further approaches, such as Linear restriction systems [257] were introduced to extend this. Others interpreted the problem by transforming the BN to a different representation, such as Weighted Boolean Function Acyclic Directed Graphs, and using best-fit algorithms to determine best assignments. To obtain approximate solutions for this, some approaches use sampling to determine an estimate of the MPE [258].

8.1.2 Model Checking

The ultimate goal of this work is to provide a supporting tool to system verification in the design phase. In recent years a promising approach to system verification is by application of the model checking process, which mainly consists of a representative model (in the given case a model of a certain functional procedure) and specifications that define nominal behavior and are used to verify the representation. The basics of Model Checking are introduced in the following based on [9].

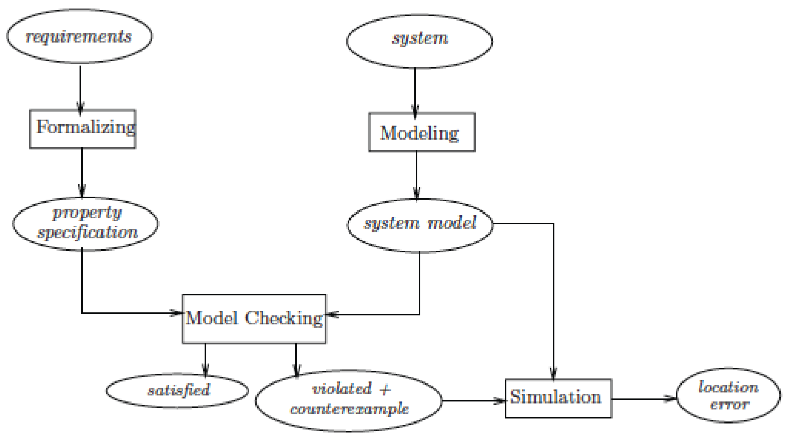


Figure 8.1: The basic model checking process is depicted and was taken from [9].

8.1.2.1 System Verification

System verification is used to verify that a system under inspection possesses certain properties, e.g. a system should never reach a deadlock. Such properties are obtained from system specifications. A system is correct if all properties, i.e. specifications, are satisfied. If any behavior, e.g. within a trace, violates a property, analysis of the malfunctioning part of the trace allows to identify locations and shape of errors.
Formal methods allow for a formalization of the system verification procedure and can according to [9] be seen as "the applied mathematics for modeling and analyzing systems, with the aim to establish system correctness with mathematical rigor". For this multiple verification techniques were developed, while the focus is on model-based verification in the following.

Model-based Verification: Such techniques use models to represent the possible system behavior. In the given case behavior corresponds to functional procedures, while models are assumed to be any model type that could have generated the observations of those procedures, which includes models such as automata or TSCBNs.
Model Checking explores all possible system states and with this checks if all properties are satisfied in the system. In general properties allow to check various behaviors including deadlocks, temporal conditions or illegal system execution. If the system encounters a violation of any property the model checker (i.e. the tool that tests properties on a model) describes an execution path that leads from initial state to a state that violates the property, which is referred to as counterexample. This counterexample can be replayed in the model to identify the malfunctioning behavior. A schematic illustration of this general verification procedure is shown in Figure 8.1. In the given case the left branch is generated with specification mining approaches. or by an expert. Further, model checking is performed by investigation of observations that are produced by the system model, as will be detailed below. In general model checkers allow to perform checking directly on traces of the system model with this verifying correctness of a trace. Hence, the right branch in the given case can be seen as the observed trace

which is input to the model checking stage, which allows to identify illegal transitions in the trace. In particular the specifications extracted with BaySpec consist of a premise and a conclusion, where the premise guarantees to only check specifications at locations where the corresponding aspect of the property and the relevant functional procedure is present. With this counterexamples refer to locations at which a certain behavior of a functional procedure was expected, but not satisfied. By knowing which property was not satisfied behavioral errors are identified.

8.1.2.2 Model Checking Process

The process of model checking in general involves a modeling, running and an analysis phase.

Modeling phase: At this stage the an accurate model of the system behavior is defined that is used as an input to the model checker. At the same time specifications are formalized using a property specification language, such as Linear Temporal Logic (LTL). in general such properties allow to check multiple types of behavior, including functional correctness, reachability, safety or liveness. In the case of BaySpec properties for functional correctness are identified only.

Running phase: In this phase the model checker is run on the system model - or in the given case on the observations of functional procedures - to check if all defined properties are valid on this model.

Analysis phase: In general, if a property is satisfied no action is required. If it is validated the counterexample is analyzed. Based on this the system model and design is refined and the property adjusted. If all properties are satisfied the system is assumed correct according to the defined properties.

8.1.2.3 Transition Systems and Traces

In general the behavior of a system under consideration (i.e. functional procedures) is modeled. For this purpose multiple models can be used, the most common of which are Transition systems TS. In the given case the observations of a trace are seen as being generated from a TS and hence, are represented as such. With this, verification of the TS is similar to verification of the trace.

Transition System: A TS is a tuple $(S, Act, \rightarrow, I, AP, L)$, with the following elements.

- S is a set of states. For functional procedures this corresponds to each state after a state change in the TSCBN.
- Act is a set of actions, which corresponds to either temporal conditions (i.e. after 0.5 seconds change state) or the direct transition τ.
- $\rightarrow$ is a transition relation from one state to another state given an action.
- $I \subseteq S$ is a set of initial states.

- $L : S \to 2^{\mathrm{AP}}$ is a labeling function that assigns a set of atomic propositions to any state s, i.e. at each state $L(s) \in 2^{\mathrm{AP}}$. In the given case the atomic propositions correspond to the observed state changes in the trace that are checked.

The TS starts in a initial state $s_0 \in I$ and evolves according to its transition relation $\to$ until no further outgoing transition is present.
Verification is now performed using the labels of states. That is, given a propositional logic formula Φ that defines a property, the state s satisfies this formula if the evaluation induced by $L(s)$ makes the formula Φ true, which is

$$s \models \Phi \text{ iff } L(s) \models \Phi \tag{8.3}$$

A simple choice is to use state names as atomic propositions, which is what is done in the given case where state names are state changes of RVs with its outcome, e.g. $A = a$.

Execution Fragment: A finite execution fragment ρ of a TS is an alternating sequence of states and actions ending with a state

$$\rho = s_0\alpha_1 s_1\alpha_2 \ldots \alpha_n s_n \tag{8.4}$$

where n is the length of the execution fragment. Thus, the execution fragment is the progression of states of the system based on actions. In the given scenario this can be seen as MSS variants of a functional procedure that are possible in the system. **Trace:** Execution fragments are system states a function may be in over time. Each of those system state has a set of atomic propositions as was introduced above. Thus, each execution fragment can be also expressed in terms of atomic propositions that were observed. For instance an execution fragment $\rho = s_0\alpha_1 s_1\alpha_2 \ldots \alpha_n s_n$ results in a trace sequence of form $L(s_0)L(s_1)\ldots$. Such sequences are called traces. As in the given scenario states and atomic propositions are assumed to be similar, a trace refers to the state changes that are observed for any functional procedure. This is formally defined as follows.
The trace of a finite path fragment $\hat{\pi} = s_0 s_1 \ldots$ is defined as

$$\mathit{trace}(\hat{\pi}) = L(s_0)L(s_1)\ldots L(s_n) \tag{8.5}$$

and thus, it the induced finite word over the alphabet 2^{AP}.

8.1.2.4 Linear-Time properties

"Linear time properties specify the traces that a transition system should exhibit" [9], i.e. defines which traces are allowed and thus, which behavior of a system is admissible.

LT Property: A LT property over the set of atomic propositions AP is a subset of $(2^{\mathrm{AP}})^\omega$, where $(2^{\mathrm{AP}})^\omega$ is the set of words that arises from the concatenation of words in 2^{AP}. In the given context this is all traces - in terms of state changes - that a functional procedure might produce. [9]

Satisfaction of LT Properties: Let P be an LT property over AP in a TS without terminal state. Then, TS satisfies P, denoted as $TS \models P$ iff $Traces(TS) \subseteq P$. State $s \in S$ satisfies P, denoted as $s \models P$ whenever $Traces(s) \subseteq P$. Hence, a TS satisfies an LT property if all its traces respect P. [9] In the given scenario, often only a subset of the Traces $\Xi \subseteq Traces(TS)$ are observed. Thus, verification is performed on Ξ.
Two ways to define LT properties and that are used within this chapter are regular expressions and linear temporal logic.

8.1.2.5 Regular Expressions

Regular expressions express a formal language to define LT properties by expressing the set of traces that are accepted. Those are introduced based on [259] in the following.
For this definition three operations *union*, *concatenation* and *closure* are defined. The *union* of two languages L and M is denoted as $L \cup M$, e.g. if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 001\}$, then $L \cup M = \{\epsilon, 001, 10, 111\}$.
The *concatenation* of L and M is the set of strings that can formed by taking any string in L and concatenating it with any string in M. For above example the concatenation is $LM = \{001, 10, 111, 001001, 10001, 111001\}$.
The *closure* of L is denoted as L^* and represents the set of strings that is formed by taking any number of strings from L, i.e. including repetitions and concatenations of those.
Each regular expression E consists of the following components and represents a language $\mathcal{L}$

- Constants ϵ and $\emptyset$ are regular expressions denoting the languages $\mathcal{L}(\epsilon) = \{\epsilon\}$ and $\mathcal{L}(\emptyset) = \{\emptyset\}$.
- If a is any symbol, then $\mathbf{a}$ is a regular expression. This expression denotes the language $\mathcal{L}(\mathbf{a}) = \{a\}$.

With this the following regular expressions is formed

- If E and F are regular expressions, then $E + F$ is a regular expression denoting the union of $\mathcal{L}(E)$ and $\mathcal{L}(F)$ which is $\mathcal{L}(E+F) = \mathcal{L}(E) \cup \mathcal{L}(F)$.
- EF is a regular expression denoting the concatenation of $\mathcal{L}(E)$ and $\mathcal{L}(F)$, i.e. $\mathcal{L}(EF) = \mathcal{L}(E)\mathcal{L}(F)$.
- E^* is a regular expression denoting the closure of $\mathcal{L}(E)$, which is $\mathcal{L}(E^*) = (\mathcal{L}(E))^*$
- (E) is also a regular expression, denoting the same language as E. Formally, $\mathcal{L}((E)) = \mathcal{L}(E)$.

8.1.2.6 Linear Temporal Logic

Another, more expressive way to represent properties, i.e. specifications, is by using LTL, which is defined in [9] in the following way. In general temporal logic can be linear, i.e. each moment in time has a single successor, or branching, i.e. each moment

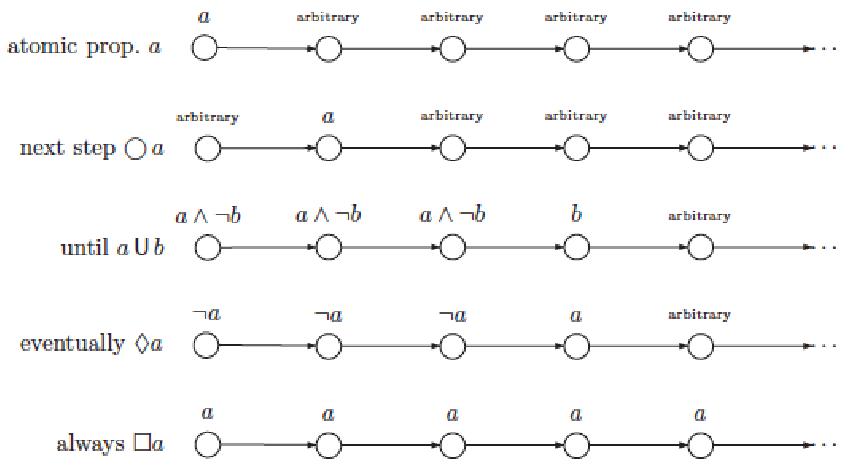


Figure 8.2: Various operators are shown. This example was taken from [9].

might have alternative successors. LTL is linear in that sense. Notably, LTL describes specifications in terms of order of events, but not exact timing of events.

LTL Syntax: LTL formulae over the set AP are formed according to

$$\phi == \text{true}|a|\phi_1 \wedge \phi_2|\neg\phi|\mathbf{X}\phi|\phi_1\mathbf{U}\phi_2$$

where $a \in$ AP. Further, operators can be derived from this, which are

$$\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \phi_2)$$
$$\phi_1 \rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$$
$$\phi_1 \leftrightarrow \phi_2 = (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$$
$$\mathbf{F}\phi = \text{true } \mathbf{U} \ \phi$$
$$\mathbf{G}\phi = \neg\mathbf{F}\neg\phi$$

, where **F** (= *eventually* or *finally*) defines that the event has to happen sometime in the future and **G** (= *globally* or *always*) defines that a property has to always hold, i.e. from now on forever. Those operators are exemplified in Figure 8.2.

8.1.2.7 Verification in MSSs

To clarify how this notation maps to the given scenario, in the following the notation is exemplified here.
The goal of verification is to check a set of specifications, i.e. properties, on a system to verify its behavior. As was stated in Chapter 2 the system behavior can be broken down into multiple functional procedures, each of which has a nominal behavior. Moreover, variants of a functional procedure are observed yielding a trace.

- **TS:** Each functional procedure corresponds to a TS. It has internal actions *Act* that transit it from a state to another state, defined with transitions $\rightarrow$. Initial states I is any state starting a functional procedure. Further, atomic propositions are state changes in the functional procedure, which is defined as a TV and its state change, e.g. for a TV A with state x an AP might be $A = x \in$ AP.

- **Execution Fragment:** An execution fragment is any variant of a functional procedure that the system might go through. For instance, when pressing a handle bar downwards the right indicator might be turned on, while pressing it upwards starts the left indicator. In this case an execution fragment might be

$$\rho = (\textit{handle bar} = \textit{default})\ \textit{user presses up} \tag{8.6}$$
$$(\textit{handle bar} = \textit{up})\ \tau \tag{8.7}$$
$$(\textit{left indicator control} = \textit{on})\ \ \textit{controller activates light} \tag{8.8}$$
$$(\textit{left indicator light} = \textit{on}) \tag{8.9}$$

, with $Act = \{$*user presses up, controller activates light* $\}$ TVs *handle bar*, *left indicator control* and *left indicator light* with its according TV states *default*, *up* or *on*. In the TS this corresponds to states $S = \{$ *handle bar* $=$ *default*, *handle bar* $=$ *up*, *left indicator control* $=$ *on*, *left indicator light* $=$ *on* $\}$.

- **Trace:** Each state in S that is observable has an AP. In the given scenario each state has the AP that describes its state, i.e. $L(s) = s$. With this a trace corresponds to an execution fragment without actions. For the above example the trace of ρ with path $\hat{\pi}$ is

$$\hat{\pi} = (\textit{handle bar} = \textit{default})\ (\textit{handle bar} = \textit{up})\ (\textit{left indicator control} = \textit{on}) \tag{8.10}$$
$$(\textit{left indicator light} = \textit{on}) \tag{8.11}$$
$$\textit{trace}(\hat{\pi}) = \hat{\pi} = (\textit{handle bar} = \textit{default})\ (\textit{handle bar} = \textit{up}) \tag{8.12}$$
$$(\textit{left indicator control} = \textit{on}) \tag{8.13}$$
$$(\textit{left indicator light} = \textit{on}) \tag{8.14}$$

- **Verification:** In the given scenario verification is performed on real traces. Thus, the exact TS is not known, but rather a subset of traces $\textit{trace}(\hat{\pi} \in T$ are considered, where T is the set of ll possible states that a system can exhibit. With this system malfunctioning is found if any observed trace has an inadmissible trace. This is verified in two ways. First, the input trace can be segmented according to functional procedures. With this, any specification that was defined for a certain functional procedure can be tested in its context. Second, if a general trace is verified, a premise can be used. With this only parts of the trace are verified where a premise, that is unique across a trace, is satisfied. The general latter case is introduced in BaySpec, while if the further is performed no premise is required.

- **Regular Expressions:** For the above example one might define that any time the handle bar changes to the *up* state the indicator should turn on. This can be defined as any trace of a functional procedure that satisfies the *regex* $(handle\ bar = default)\ ?\ (handle\ bar = up)\ \ (left\ indicator\ control = on)\ ?$ $(left\ indicator\ light = on)$, where optional intermediate states are allowed.

- **LTL Properties:** The above regular expression is more expressively defined by stating that once the handle bar changes to the up state eventually the left light should turn on, which has to always hold across the whole trace. This is defined by specifying a property

$$\mathbf{G}((handle\ bar = up)\ \wedge \mathbf{F}(left\ indicator\ light = on)) \tag{8.15}$$

  The proposed approach BaySpec is used to extract specifications of this type.

8.2 Most Likely Behavior of MSSs

The focus is on two approaches that aim to identify the dominant states and its most important variants in a trained TSCBN. For this existing approaches of MPE are discussed. Next, TSCBNs capture multiple variants of a functional procedure and thus, allows to compare observed cases of MSSs to find Specifications of appropriate strictness. The approach BaySpec is presented here in the following, as it is used as a method for the final building block of the DM Pipeline for Specification Mining.

Most Probable Explanation in TSCBNs TSCBN model distributions of functional procedures in both state and time. Further, the model is overfitted intentionally on the observed data. Thus, obtaining the k most probable constellations of a TSCBNs is similar to obtaining the MSSs that were predominantly observed. In the context of diagnosis this can be seen twofold. First, assuming that the training data contained mainly correct data and little noise this behavior is the nominal behavior. Second, assuming that the training data is faulty data the observed behavior is faulty. In the further case, experts can use the information to improve system knowledge and to design specifications based on the template given by the dominant MPE MSS. In the latter case, the symptoms of the error are represented within the functional procedure, which allows to learn a knowledge base of errors. This base could be used to detect such then, known errors and to initiate fixes.
As discussed in Section 8.2 multiple methods for MPE estimation exist. However, a simple and effective solution for this is found by sampling from the TSCBN. That is, assuming to have drawn sufficiently many constellations of the TSCBN the resulting distribution indicates the dominant states. For this the following is done.

1. From the targeted TSCBN remove all temporal nodes T and edges leading to those.
2. Sample N_{sam} of times from the remaining State model of the TSCBN, e.g. using Gibbs sampling. For the i-th sampling an outcome $\bar{v}_i$, consisting of values for all RVs of the network, is found. The set of all N_{sam} outcomes $\bar{v}_i$ is $\bar{V}$.
3. The frequency of all unique outcomes in $\bar{V}$ is counted resulting in a distribution that maps a unique outcome combination $\hat{v}$ to its frequency $f(\hat{v})$.

4. Next, by exploiting the TSCBN structure (i.e. that temporal nodes are leaf nodes) and assuming Gaussian distributions per temporal node, for each defined conditioning on any temporal node Δt_{ij} a parameter μ and Var is given after parameter estimation. This value of Δt_{ij} (under the corresponding conditions) is used as estimate of the temporal behavior. With this, a full MPE estimate is found in state and in time.

It is assumed that the k most frequently observed combinations $\hat{v}$ are the k MPEs of the TSCBN in state and time. Those entries are used by the expert within the further process.

8.3 Automated Specification Mining using BaySpec

Next to the extraction of dominant behavior the main focus of this work is on extracting specifications from large-scale distributed systems. By performing the previous steps of the proposed DM pipeline a representation of relevant TVs and functional procedures was identified. With this, a TSCBN model is found per functional procedure that contains potentially nominal behavior. However, this nominal behavior cannot directly be used as specification as it would be to strict and would indicate an error for any model constellation that is not exactly as defined by the model. Thus, a strategy needs to be found that allows to find meaningful Specifications of appropriate strictness. An approach to this was presented by us in [10] and is summarized in the following. Notably, this approach focuses on the state model of the TSCBN only and thus, is able to deduce specifications in terms of symbol order rather than including timing of the specifications.

Challenges: Main challenges for the extraction of specifications from a TSCBN include the following.

- **Valid LTL:** TSCBNs represent probabilistic representations of MSSs as a multidimensional system, rather than as a sequence of events, as it is the case in automata for example. This requires to identify path representations from the network to enable identification of specifications in LTL semantics.
- **Strictness:** On the one hand specifications that do represent the exact observed behavior are often overfitted on the observed data, i.e. define the specification such that it is violated as soon as only a single literal is different than the observation. Thus, for any tested data set noise or correct alternations would be identified as potential error, yielding a big result set of found violations. This makes it meaningless for an expert, as a manual post processing would be required. On the other hand learned Specifications might be also defined too blurry, which leads to those specifications never firing, making those again meaningless. Thus, the challenge is to find specifications of right strictness. BaySpec solves this by performing merging operations.
- **Complexity:** Growing sizes of TSCBNs result in multiple variants of paths that have to be inspected. This increasing search space requires efficient methods for inspection of those paths.

- **Noise:** Multiple variants of MSS behavior are represented in a TSCBN, while only a subset of those represents nominal behavior rather than noise. Further, in case noise was learned as having high likelihood in the TSCBN, comparable TSCBNs that do not contain noise or softening strategies within a TSCBN can be used to still find valid LTL specifications that compensate for this noise, e.g. by making it optional.

8.3.1 Overview

The overall procedure consists of the following main steps.

1. **Network Pruning:** From the targeted TSCBN remove all temporal nodes T and edges leading to those.
2. **Mining Graph Conversion:** Convert the resulting state model into a Mining Graph that represents connections in terms of RVs with its values rather than in terms of the RVs only. Then, assign weights within the Mining Graph based on the conditional probability distributions of the state RVs in the TSCBN.
3. **Candidate Search:** Use a shortest path search to find paths of highest likelihood as potential Specification candidates. Those candidates are in general too strict and thus, meaningless.
4. **Path Merging:** Merge similar Specification candidates until appropriate strictness is found or discard if those are meaningless. This results in a set of specifications which are represented as regex. This step can be done in two ways either based on metrics or by using a validation TSCBN.
5. **LTL Conversion:** Convert all regex expressions to LTL Specifications that are used for verification of incoming traces.

Per functional procedure LTL Specifications result, which were found in a semi-automated manner by including expert input during model generation. This concludes the DM pipeline. In the following the main steps are described.

8.3.2 Mining Graph Conversion

A TSCBN captures a functional procedure in terms of temporal and causal dependencies. At this stage only the state model of the network is considered, as it is enough to capture the observed MSS behavior in terms of order. Thus, with causality between state changes represented as edges, any likely path through the state model represents a behavior that was dominantly observed. For this it is assumed that this behavior corresponds to nominal behavior, while noise was observed less and thus, noisy behavior corresponds to less likely paths. With this, a likely path with strong causal dependencies (i.e. high conditional probabilities) among its nodes is likely to represent one variant or aspect of a functional procedure which is interpreted as a candidate specification for this functional procedure. In contrast to process models, which allow to extract paths of sequential behavior as the MSSs are considered as event sequence, this allows to exploit

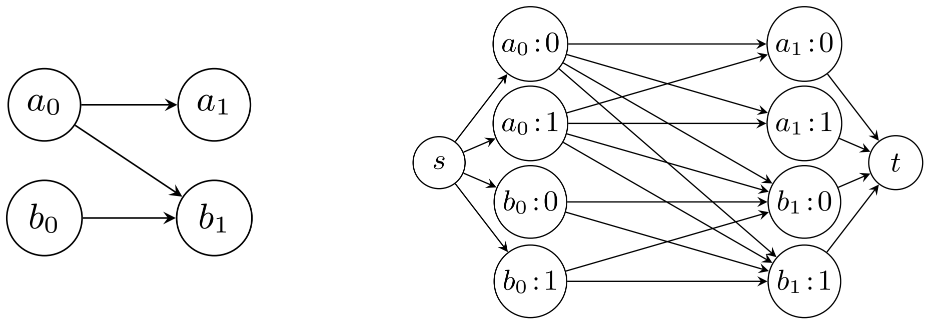


Figure 8.3: A Bayesian Network with two states 0 and 1 per node is shown on the left and the resulting *Mining Graph* after conversion is shown on the right. This figure was taken from [10].

temporal causality and structure that is represented in the model. This makes it less prone to find false positive specifications and to be more expressive.

Path Search Inference in TSCBNs: Existing inference algorithms of BNs aim to find the MPE or the partial MAP as described above. That is, those try to identify the configuration of the TSCBN with best explanation. Other algorithms for finding most likely paths, e.g. the Viterbi algorithm [260], are defined for one parent per node only and hence cannot find paths in BNs with multiple parents.
However, the aim is to identify sequential paths in this multidimensional MSS representation of a TSCBN. To solve this, the state model of the TSCBN is transformed into a weighted graph that is denoted as a *Mining Graph (MG)* (see Figure 8.3). This representation is obtained by marginalizing out parent-nodes per candidate path. This path is searched for likely paths, which form candidate specifications, which is solved by BaySpec as presented in this section.

Definition of Mining Graph: A Mining Graph is defined as follows. Given a BN (G, Θ), with graph G and parameters Θ over RVs $\mathbf{X}$, a Mining Graph is a DAG $(\bar{V}, \bar{E})$, with

$$\bar{V} = \left(\bigcup_{i=1}^{|\mathbf{X}|} \bigcup_{j=1}^{|\Omega_{X_i}|} \vartheta_{ij} \right) \cup \{s, t\}$$

$$E^{(X)} = \big\{ (\vartheta_{im}, \vartheta_{jn}) \mid \vartheta_{im}, \vartheta_{jn} \in \bar{V} \wedge (X_i, X_j) \in E \big\}$$

$$E^{(s)} = \big\{ (s, \vartheta_{im}) \bigm| \vartheta_{im} \in \bar{V}, X_i \text{ is the first instance of a RV} \big\}$$

$$E^{(t)} = \big\{ (s, \vartheta_{im}) \bigm| \vartheta_{im} \in \bar{V}, X_i \text{ is the last (not first) instance of a RV} \big\}$$

$$\bar{E} = E^{(X)} \cup E^{(s)} \cup E^{(t)}$$

The set of vertices $\bar{V}$ consists of a vertex for each discrete value a RV $X_i \in \mathbf{X}$ can take and two additional vertices s and t representing an artificial start and terminal vertex, respectively. s and t are required as those define the start and target nodes for the shortest path search. The set of edges $\bar{E}$ contains an edge between two vertices from $\bar{V}$

if their corresponding RVs X_i and X_j are connected in the BN. Vertex s is connected to all vertices of initial RVs per TV (i.e. RVs with index 0). Vertex t is connected with all vertices whose RVs is the last instance per TV. To avoid paths with a single vertex, vertices of a single-instance RV are not connected to t, prohibiting paths of length one, such as (s, ϑ_{ij}, t).

Edge weights and Marginalization: As can be seen in Figure 8.3, per edge in the MG a unique weight value is defined that represents the likelihood of to subsequent state changes of the TSCBN to occur. In TSCBNs however, those weights are implicitly given in terms of conditional probability distributions of a state node given by multiple parents. To thus, be able to perform a meaningful search on the MG appropriate weights need to be found. This is done by marginalization of parent RVs at each vertex in the TSCBN in order to find edge weights to this vertex. Formally this is described as follows. For a target node X_j of an edge (X_i, X_j), let
$Y\left(e = (\vartheta_{im}, \vartheta_{jn})\right) = \{Y_1, \dots, Y_k\} \coloneqq \mathrm{Par}(X_j) \setminus \{X_i\}$ be the set of all parent RVs without source node's RV X_i. The conditional probability $P(X_j \mid X_i)$ is defined as

$$P(X_j|X_i)=\sum_{Y_1}\cdots\sum_{Y_k}\left(P(X_j|X_i,Y_1,\dots,Y_k)\cdot P(Y_1)\cdot\ldots\cdot P(Y_k)\right). \tag{8.16}$$

Let $\mathrm{Par}(X_i) \coloneqq \{R_1, \dots, R_p\}$. The marginal probability of $P(X_i)$ is defined by

$$P(X_i)=\sum_{R_1}\cdots\sum_{R_p}\left(P(X_i|R_1,\dots,R_p)\cdot P(R_1)\cdot\ldots\cdot P(R_p)\right). \tag{8.17}$$

The function w assigns a weight to each edge and is defined by

$$w(e=(\vartheta_{im},\vartheta_{jn}))=\begin{cases}0 & \text{if } \vartheta_{jn} = t\\ 0 & \text{if } \vartheta_{im} = s\\ 1 - P\left(X_j = x_{jn} \mid X_i = x_{im}\right) & \text{otherwise}\end{cases} \tag{8.18}$$

With this, the edge weight between a node ϑ_{im} (denoting RV $X_i = x_{im} \in \Omega_{X_i}$) and a node ϑ_{jn} is $1 - P\left(X_j = x_{jn} \mid X_i = x_{im}\right)$. If X_j has further parents besides X_i, the probability $P\left(X_j = x_{jn} \mid X_i = x_{im}\right)$ is calculated by marginalization using (8.16) which uses (8.17) to calculate the marginal probability. Edges to t and from s have weight 0. With this, a full representation of a TSCBN as a weighted MG is found, which is used in the following to extract meaningful LTL specifications.

8.3.3 Candidate Search

The goal of this step is to find paths of highest likelihood as potential Specification candidates. This is achieved in the following manner. It is assumed that most likely paths in the BN are desired and corresponds to potential specifications. Those likely paths are found in terms of maximum products or averages. In the given case, the aim is to find all most probable paths in the MG that exhibit a maximum average likelihood $p_{\min}$ (i.e. a minimum average edge weight $w_{\max} = 1 - p_{\min}$).
Existing k-shortest path search algorithms look for paths with minimum length by considering summation or products of edge weights. In case of BNs using summations would

Algorithm 3

Minimum Average Edge Weight Path (modified Dijkstra) from [10]

Input: *graph G, start node ϑ_s, target node ϑ_t, root path π_r*

Output: *minimum average edge weight path from ϑ_s to ϑ_t given π_r*

```
$V^* =$ reachable($G$,start)
$V^*$.topologicalSort( )
for each vertex $v$ in $V^*$ do
    distance[$v$] = new map(default = infinity)
    previous[$v$] = new map(default = undefined)
end for
distance[$\vartheta_s$][$\pi_r$.edges] = $\pi_r$.probability
while not $V^*$.isEmpty( ) do
    $u = V^*$.pop(0)
    for each neighbor $v$ of $u$ do
        for each edges, avg in distance[$u$] do
            if $u$ is $s$ or $v$ is $t$ then
                newAverage = distance[$u$][edges]
                edges = edges−1
            else
                newAverage = (edges $\cdot$ avg + weight($u, v$)) / (edges+1)
            end if
            if distance[$v$][edges+1] > newAverage then
                distance[$v$][edges+1] = newAverage
                previous[$v$][edges+1] = $u$
            end if
        end for
    end for
end while
return previousToPath(previous, $\vartheta_s, \vartheta_t$)
```

prefer shorter paths, as those include less weights and are thus smaller. The same holds when using products, as shorter paths are multiplied less often than longer ones and weights are between 0 and 1. This would limit the length of the candidate specifications, which is not intended here as the aim is to support mining of specifications of arbitrary lengths. Using averages however, does treat paths of different lengths equally, especially when the constraint of a minimum path length is defined.
However, no existing approach supported finding paths of minimum average weights but rather shortest paths.

Extended Yen Search: Therefore, an algorithm is introduced to solve this using an extension of Yen's search [261]. The extension consists of two main improvements. First, path search is stopped if a metric of a path falls below a given threshold. Second, in its kernel, Yen uses the modified Dijkstra algorithm (listed in Alg. 3) to find paths with minimum average weights for a given start and end node. This modification is required, as when searching minimum average paths, greedy algorithms, e.g. Dijkstra's algorithm, can result in non-optimal solutions, e.g. in Figure 8.4 taking a local optimum would not lead to an optimal solution. For b the path with minimum average path weight is (a, b). But, the globally best path depends on the number of edges taken so far and the edge weights to come, so that the path with minimum average edge weight is (a, x, b, c). This is solved in the algorithm by remembering for each vertex the best path for every number of edges reaching that vertex (initialization in lines 4-7). A further loop is added that not only iterates over all neighbors of a vertex u but also over all best paths with a different

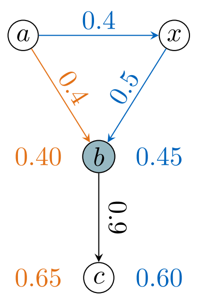


Figure 8.4: The limitations of the Dijkstra Algorithm are illustrated [10].

number of edges reaching u (line 11). The new average path weight is computed in lines 12-17 where edges including vertices s and t have to be handled separately as those nodes are neglected.
This results in a candidate set of likely paths in the TSCBN, which resemble a set of strict temporal properties of the system. Those properties are initially represented as regular expressions (*regex*), as this representation is well suited to allow for both merging and for measuring of strictness at the right level of abstraction required here. For this any path $\pi = (\vartheta_1, \vartheta_2 \ldots, \vartheta_n)$ is represented as a *regex* $\vartheta_1 \vartheta_2 \ldots \vartheta_n$ of n single literals. In case of TSCBNs each literal resembles a RV and its value (e.g. $A_0 = 1$). However, within the further scope such path elements are denote as a single symbol for better readability. This translation operation of a path to its *regex* is done in line 4.
Further, for the purpose of merging the full representational capacity of the *regex* is not required, as here only finite paths occur. Thus, a reduced syntax is used, which only includes the ?-quantifier and the alternative notation, where e.g. (a — b)c, matches either a path *ac* or *bc*.

8.3.4 Path Merging

The resulting candidates extracted during path search express likely behavior of the observed functional procedure. However, those specifications are meaningless as those are too strict. That is why loosening of those specifications needs to be performed. This is basically done by merging similar candidate specifications. For this two merging approaches were proposed, that successively loosens candidate specifications until good strictness is reached, i.e. the resulting specification is strict enough and not too trivial or a tautology. This is, a metric based approach and a comparison based approach which are introduced in the following.

8.3.4.1 Metric Based Merging

The first approach is based on metrics only and is shown in Alg. 4. There, the basic idea is to find most similar paths (from the set of found paths) and iteratively merge those until looser specifications are obtained. After each merging metrics such as *literal ratio*, *combination count* and *conformity* are used to measure how strict a merged specification is. This is repeated until specifications are found that lie within a defined strictness

Algorithm 4
Metric based approach from [10]
Input: *paths P, target range*
Output: *specifications s*

```
$D$ = editDistances($P$)
$s$ = []
for each path $p_i \in P$ do
    $r = p_i$.toRegex( )
    for each path $q \in P$ sorted by $D_{i-}$ do
        $r$.merge($q$)
        if $r$.isRedundant($s$) then
            break
        end if
        if metrics in target range then
            $s$.removeRedundantRules($r$)
            $s$.add($r$)
            break
        else if any metric surpasses target range then
            break
        end if
    end for
end for
return $s$
```

range. This is described in more detail in the following.

Basics: Within a set of candidate specifications it is important to merge related, i.e. similar, candidates only, as merging completely distinct properties results in tautologies making those meaningless. To be able to determine such similarity a measure is required. Common measures for the distance of two candidate specifications are edit distances, that measure the distance in terms of modifications between two sequences. A common edit distance is the Levenshtein distance and a representation for comparing paths is the Levenshtein matrix. Those terms are defined in the following.

Levenshtein matrix [188]: The Levenshtein matrix $M^{\pi_a,\pi_b} \in \mathbb{N}^{m\times n}$ for two paths $\pi_a = (a_1,\ldots,a_m)$ and $\pi_b = (b_1,\ldots,b_n)$ is recursively defined by

$$M^{\pi_a,\pi_b}_{i0} = \textstyle\sum_{k=1}^{i} w_{\text{del}}\,, \qquad M^{\pi_a,\pi_b}_{0j} = \textstyle\sum_{k=1}^{j} w_{\text{ins}}\,,$$

$$M^{\pi_a,\pi_b}_{ij} = \begin{cases} M^{\pi_a,\pi_b}_{i-1j-1} & \text{if } a_i = b_j\,, \\ \min\begin{cases} M^{\pi_a,\pi_b}_{i-1j} + w_{\text{del}} \\ M^{\pi_a,\pi_b}_{ij-1} + w_{\text{ins}} \\ M^{\pi_a,\pi_b}_{i-1j-1} + w_{\text{sub}} \end{cases} & \text{otherwise}\,, \end{cases}$$

where w_{del}, w_{ins} and w_{sub} are weighted costs for deleting, inserting and substituting a symbol. In the investigated approach $w_{\text{del}} = w_{\text{ins}} = w_{\text{sub}} = 1$.

Levenshtein distance [188]: The Levenshtein distance $L(\pi_a, \pi_b)$ between two paths $\pi_a = (a_1,\ldots,a_m)$ and $\pi_b = (b_1,\ldots,b_n)$ is given by M^{π_a,π_b}_{mn} where M^{π_a,π_b} is the Leven-

	literal a	?-quantifier a?	alternation (a — b)
delete	a?	a?	(a | b)?
substitute with c	(a | c)	(a | c)?	(a | b | c)
insert c (at end)	a c?	a? c?	(a | b) c?

Table 8.1: Edit operation rules for regular expressions as introduced in [10].

shtein matrix for π_a and π_b.

i) Calculating edit distances between paths: To ensure more similar paths to be merged first, similarity between paths is computed in line 1 of Alg. 4. The order in which paths q are merged with a starting path p is determined by the Levenshtein distance between them. If two paths have the same distance to a starting path, the path with the higher average probability is merged first. All pairwise computed Levenshtein distances are stored in a symmetrical matrix D.
For a set of paths $P = \{\pi_1, \ldots, \pi_m\}$ the edit distance matrix $D \in \mathbb{N}^{m \times m}$ is defined by its elements

$$(D_{ij})_{0 \leq i,j \leq m} = L(\pi_i, \pi_j) \tag{8.19}$$

Also, the minimal number of editing operations (insertion, deletion, substitution) that are required to transform one path into the other is called the *minimum edit distance*.

ii) Merging paths: So far specifications are defined as *regex*. Those are merged by the edit operations *delete*, *substitute* and *insert* to give new *regex*s matching at least both input expressions (line 6). With this successively softer expressions are found, which are gradually decreasing in strictness as only similar specifications are repeatedly merged. With this, it is possible to stop merging if good strictness is reached.
Merging is done according to a defined list of rules which loosens the expression, that is shown in Table 8.1. By comparing the Levenshtein matrix of the two expressions those rules are successively applied to the expressions such that both input expressions are represented in the resulting merged expression.

- **Merging operations:** As shown in Table 8.1 merging of expressions includes the following operations. That is, deletion and insertion each modify the expression due to the fact that a literal is present in one but not in the other input expression. Thus, the missing or additional literal is added as optional, e.g. in the table this is $a?$ for deletion and $c?$ for insertion. Substitution means that at a point where previously one literal was observed (e.g. a) in the expression another literal is present in the compared expression (e.g. c). Thus, the literal might be either the one or the other after loosening, e.g. in the table $a|c$. With this, deleting any literal a gives the optional literal $a?$, $a?$ gives $a?$ and $(a|b)$ gives $(a|b)?$. Substitution of a with c yields $(a|c)$, of $a?$ yields $(a|c)?$ and of $(a|b)$ gives $(a|b|c)$. Lastly, insertion of c to a gives $ac?$, to $a?$ gives $a?c?$ and to $(a|b)$ gives $(a|b)c?$.

		a	c	D
	0	1	2	3
a	1	0	1	2
b	2	1	1	2
c	3	2	1	2
d	4	3	2	2

Table 8.2: The Levenshtein matrix for two expressions *abcd* and *acD* is shown as given in [10].

- **Order of merging:** The order of edit operations is derived using the Levenshtein matrix of the two expressions. For this the order in which to execute the edit operations is precomputed using a *backtracking* algorithm, which searches a non-increasing path within the Levenshtein matrix M from M_{mn} to M_{00}. This is performed during merging in line 6. Thus, the edit operation list for two paths π_1 and π_2 is computed as follows. Given the algorithm is currently at M_{ij}, for a step to the left an insertion of symbol π_{2_j} at position i is added, for a step upwards a deletion at position i is is added and for a diagonal step to the upper left, where the value at the upper left is lower, a substitution with symbol π_{2_j} at position i is added to the list. The operation positions refer to positions in the path π_1. For a diagonal step to the upper left, where the value at the upper left is not lower, no entry is added to the operation list, as then symbols are equal.

After computing the edit operations, the two expressions are merged by executing the corresponding merge operations defined in Table 8.1, e.g., the paths $\pi_1 = (a, b, c, d)$ and $\pi_2 = (a, c, D)$ are transformed into the *regex*s *abcd* and *acD* respectively. Merging them results in the new *regex* $a\,b?\,c\,(d\,|\,D)$ by applying the rules for delete and substitute from Table 8.1. Table 8.2 shows the Levenshtein matrix for those expressions with the highlighted backtracking path.

iii) Metrics: As mentioned before, in the metric based approach the merging of ii) is performed until the resulting expression is within a strictness range defined by the following three parameters.

- **Literal Ratio:** This measure r_L is defined as the ration between the number of single literal characters n_s and the total number of symbols n_t in the expression. That is,

$$r_L = \frac{n_s}{n_t} \tag{8.20}$$

  There a single literal refers to a literal that has neither a ? nor a | quantifier. For instance in an expression $ab?c(d|e)$, two single literals are given with a and c, while all symbols include a, $b?$, c and $(d|e)$. This gives values $n_s = 2$ and $n_t = 4$ resulting in the literal ratio $r_L = \frac{2}{4} = 0.5$. A high value for r_L indicates many single literals and thus, a very high degree of strictness as anything violating this exact order of single literals violates the specification. In contrast to that, a low value of r_L

yields many alternating and ? quantifiers, which results in very low strictness, as for the extreme case where only ? quantifiers are given any input is allowed and thus, the specification is always satisfied. Consequently, a balanced value needs to be found here.

- **Combination Count:** The combination count r_C is the total number of paths in the TSCBN that match the current specification. For instance in a *regex* $a\,b?\,c\,(d\,|\,D)$, matching paths include *abcd*, *abcD*, *acd* and *acD*. This results in a combination count $r_C = 4$. A r_C that is high in relation to the total number of possible paths in the TSCBN indicates blurriness, while a very low number indicates high strictness. Consequently, a balanced value needs to be found here, too.

Merging is done iteratively for all found candidate paths that are pairwise merged with its most similar paths. Per iteration, if both metrics fall within defined target thresholds, the merging operation is stopped and the loosened *regex* is added to the result set (line 10). If the acceptance window is not reached the merged candidate specification is dropped (line 14).
Lastly, BaySpec removes trailing and leading symbols with a ?-quantifier, as those do not add relevant information to the expression. With this, for instance the expression $x?\,a\,(b\,|\,B)\,c\,(y\,|\,Y)?$ will be reduced to the expression $a\,(b\,|\,B)\,c$.

iv) Conversion to LTL: *regex*s can be well used as representation to perform merging operations for softening, but have some practical limitations which is why a conversion to an LTL specification is performed at this step. LTL has the following advantages over regex. First, LTL allows to define premises which are applicable when the specification is to be tested on the trace. If this was not the case a lot of false positive violations of any LTL specification would occur. Second, LTL is more expressive as it allows to define chains in more detail, e.g. by distinguishing operators such as next, until and finally. Third, LTL is more commonly used in model checking tools.
That is why, all extracted *regex*s are converted to LTL *formula*s of events, premises and conclusions. This formula has the following format.

$$\boxed{\begin{aligned} \textit{formula} &:= \mathrm{G}(\textit{premise}) \\ \textit{premise} &:= \textit{event} \rightarrow \mathrm{XG}(\textit{premise}) \text{ — } \textit{event} \rightarrow \mathrm{X}(\textit{conclusion}) \\ &\quad \text{— } \textit{event} \rightarrow \mathrm{XF}(\textit{conclusion}) \\ \textit{conclusion} &:= \textit{event} \text{ — } \textit{event} \wedge \mathrm{X}(\textit{conclusion}) \\ &\quad \text{— } \textit{event} \wedge \mathrm{XF}(\textit{conclusion}) \text{ — } \textit{event}\, \mathrm{U}\, (\textit{conclusion}) \end{aligned}}$$

As described above, this formula specifies that if the premise is found in the trace, the conclusion must follow. With this, unique occurrences of a premise are found such that the specification only fires if the shortest unique premise is found. With this the above symbols are defined as follows.

- **Event:** Any single literals and alternations that are represented in the *regex* are translated to an event in LTL. An event represents a symbol of the *regex.* Any single literal of the regex is translated to a single event, while for an alternation the event is the disjunction of the alternation's literals. For instance a regex $(a \mid b \mid c)$ would be converted to the event $(a \vee b \vee c)$.

- **Premise:** Expressions in LTL should only match at locations where the expected behavior is present. To ensure this a premise is used, that consists of a prefix of symbols from the *regex* that is unique within all combinations. With this, when applying an LTL specification to a trace the LTL check is only performed if the premise is given.
- **Conclusion:** As the premise already contains a subset of behavior that is to be tested, the conclusion is comprised of all remaining symbols of the *regex* that are not part of the premise.

As LTL is more expressive than *regex* additional information is required for the conversion process. That is, in order to convert parts of the LTL formula to a representation of appropriate strictness an according symbol needs to be found between two symbols of the regex, i.e. between two path elements. For this optionally statistical information is recorded, e.g. during learning of the BN a histogram of numbers of events that occurred between two nodes is counted and stored per edge. More precisely, it is stored how many times in the training trace the two events (= RV and its states) occurred directly one after the other, with distance two, distance three and so on. Based on this histogram any edge within a path can be assessed, e.g. if event A was always followed directly by another event B or if it sometimes occurred after two and sometimes after three events defines if a next or a finally operator is required between the two symbols of the *regex*. The following rules are used by BaySpec for this.

- **Sub sequence:** As stated before events in a regex (e.g. ab) can either result in a "next" X, implying b to appear immediately next, or in a "finally" XF operator, which implies b to appear eventually in the future (XF is used to allow two identical successive events which is not possible by using F only.). To distinguish those two cases the histogram is used. The conversion "next" X is used if the number of intermediate traces makes up a higher percentage than a threshold, i.e. if the majority of events supporting this expression has these events directly follow each other. If this is not the case the events are not directly consecutive, which is called *intermediate noise* between the events. That is, for events with intermediate noise the conversion to the "finally" XF operator is performed.
- **?-Quantifier:** Symbols with this quantifier are optional, e.g. $a?b$. Those elements can either occur or not occur. BaySpec translates those elements into the "until" U operator (e.g. $a\,U\,b$), which is used if always an a occurs until a b occurs. That is, in the histogram between any two symbols, the number of intermediate noise events between all as until a b occurs have to be zero. If any intermediate noise is seen, the ?-quantified symbol is omitted during conversion, e.g. resulting in b for the expression $a?b$.
- **Multiple ?-Quantifiers:** Multiple consecutive ?-quantified symbols, e.g. $a?\,b?\,c?\,d$, are translated recursively using the rule defined for the ?-quantifier. That is, starting from the last symbol (here d) it is recursively tested if the preceding ?-quantified

symbol has *intermediate noise*. If no noise is present, the corresponding symbol is translated to the "until" operator U. Else, all ?-quantified symbols left to the current symbol are dropped, e.g., given $(a?, b?)$ as the only pair with *intermediate noise* in $a?\,b?\,c?\,d$ results in the LTL formula $b\,U\,c\,U\,d$.

v) Removal of redundant specifications: To reduce the number of redundant specifications post processing is performed. That is, after any merging operation resulting expressions might be either meaningless or already satisfied by an already mined specification. To prevent the further, measures are used to drop the specification. For the latter, it is defined, that as soon as the current specification is the result of merging paths that already made up another found specification, the current iteration is terminated and the specification discarded (lines 7 and 11).

8.3.4.2 Comparison Based Merging

Using only metrics of strictness includes a measure of how blurry a found expression is. However, in the context of multiple involved instances that are observed well strictness is also defined as validity of an expression among all instances producing the observations. Such information is excluded in the metric based approach and is overcome in the comparison based approach which is revised here.
Excluding information gained among instances results in the algorithm merging longer than required. However, when considering multiple instances that produce the same functional procedure a valid specification is found that holds for both instances. For example, such differences might result from system modifications during development, different environmental conditions during recording or instances of different type (e.g. two models of cars). To include the variations among functional procedures between the two instances, similar functional procedures are used to learn a TSCBN per instance. Based on structural variations among those networks the strictness of the resulting specification is extracted.
That is, BaySpec performs the same steps as in the metric based approach. However, in addition to using metrics as stopping criterion, the second model, that is called validation model, is used to decide when to stop merging paths. It is assumed that specifications that are correct should be likely in both TSCBNs, as those should represent the intended behavior of both instances. With this, structural noise is filtered during merging, resulting in specifications of good strictness in both TSCBNs. For this, during merging, each path is checked on the validation model and accepted if a minimum average likelihood is found on it. With this, the resulting algorithm is identical to Alg. 4, except that after line 2 $K_{\text{val}} = A_{\text{val}}$.toKripke() is inserted and lines 5 to 17 are replaced by the following code, with validation BN A_{val} as additional parameter.

```
1: for each path $q \in P$ sorted by $D_{i-}$ do
2:     $r$.merge($q$)
3:     if $r$.isRedundant($s$) then
4:         break
5:     end if
6:     if any metric surpasses threshold then
7:         break
8:     end if
```

```
 9:     if ModelCheck($r, K_{\text{val}}$) then
10:         $s$.removeRedundantRules($r$)
11:         $s$.add($r$)
12:         break
13:     end if
14: end for
```

Comparison-based stop: After each merging model checking is used on the validation model to decide if it is valid or not. If validation is positive, merging continues, while if this is not the case the algorithm stops. To perform this checking operation of *regex* of merged paths on the validation model, model checking tools can be used. To allow for this, the TSCBN structure is converted in any automaton required and the regex represented in the according language.

Implementation details: Within the scope of this work the following implementations are used for this step. In the implementation in [10], the validation model is transformed into a Kripke structure. This is done similar as in the LTL conversion, where the ?-quantified symbols in front and at the end of the *regex* are removed before translation resulting in the following shape:

$$\begin{aligned} \mathit{formula} &\coloneqq \mathit{event} \wedge \mathrm{EX}(\mathit{tail}) \\ \mathit{tail} &\coloneqq \mathit{event} — \mathit{event} \wedge \mathrm{EX}(\mathit{tail}) — \mathrm{E}(\mathit{event}\,\mathrm{U}\,(\mathit{tail})) \end{aligned}$$

The formula specifies a chain of events that should hold in the validation model and Events are defined similar to the LTL case. Unlike before, no ?-quantified symbol is dismissed but rather translated into an until operator as explained in the LTL case. E.g., *abc* is translated into $a \wedge EX(b \wedge EX\,c)$.
The Mining Graph of the validation model is transformed into a Kripke structure (S, S_0, R, L), where vertices become states in S and edges become elements in the transition relation $R \subseteq S \times S$. Additionally, a loop (t, t) at the terminal node is added as the transition relation is required to be left-total, i.e. $\forall s \in S \exists s'$s.t.$(s, s') \in R$. The set S_0 of initial states contains the artificial start node s. The labeling function L defines a label $X = \omega$ for each state, indicating the random variable X has the value ω.
As a model checker the pyModelChecking package[1] is used which in written in Python. The model checker returns all states of the Kripke structure which satisfy the CTL formula. If the returned set is non-empty the merging loop is stopped as the mined specification is not valid in both BNs. The found specification is only accepted if at least one path in the validation model matches the *regex* formula and exhibits the same minimum likelihood. Lastly, besides structural checks a metric based check is performed as well. If the literal ratio falls below or the combination count is above a threshold, merging is stopped to avoid specifications that are too soft.

8.4 Evaluation

The aim of this work is to perform Specification Mining on traces of large scale distributed systems. For this a TSCBN representation for each functional procedure is

[1] `https://github.com/albertocasagrande/pyModelChecking`

found. In [10] it was shown that BaySpec is well suited to extract meaningful specifications from such representations, as it is able to generate less False Positives, as well as more expressive specifications. This was presented in a synthetic evaluation, which is recapitulated in the first part of this section. Further, it was shown that BaySpec can do this also for real world models, which will be revised here as well. With this, it is validated that BaySpec is well suited as the last step of the DM pipeline.
In addition to that, in a case study in [8] it was shown how the process of TSCBN learning together with MPE and BaySpec is used to extract meaningful information from automotive trace data. This work will be recapitulated in the last part of this section and shows how the last two steps of the proposed pipeline perform in combination with the two investigated inference methods.

8.4.1 Synthetic Evaluation of Performance and Expressiveness of BaySpec

8.4.1.1 Setup

Environment: Experiments are conducted on a Lenovo^{TM} T480s equipped with two Intel® Core® i5-8350U 1.70GHz CPUs with 16 GB of RAM. Further, BaySpec is run without the full statistical refinement of the histograms, but rather only distinguishes next and finally, as well as it includes the until.

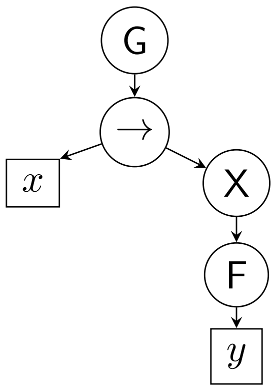


Figure 8.5: The syntax tree of the LTL formula $\mathrm{G}(x \to \mathrm{XF}\, y)$ is shown to exemplify the notion of complexity.

Metrics: Expressiveness is measured in terms of *Complexity* of learned specifications, while the *False Positive (FP) rate* defines the precision, which were defined as follows.

- *Complexity:* This value is measured in terms of the height and unique event count of the resulting LTL formula's syntax tree. The height is the number of nodes between the root and the deepest leaf. The unique event count is the number of leaves with different events. An example of this is shown in Figure 8.5. The frequency of specifications of certain complexity is measured. More frequent specifications of higher complexity are desired, as this corresponds to better expressiveness.
- *FP Rate:* The *False-Positive (FP) Rate* is the percentage of specifications that are falsely learned. In the given scenario this corresponds to specifications that

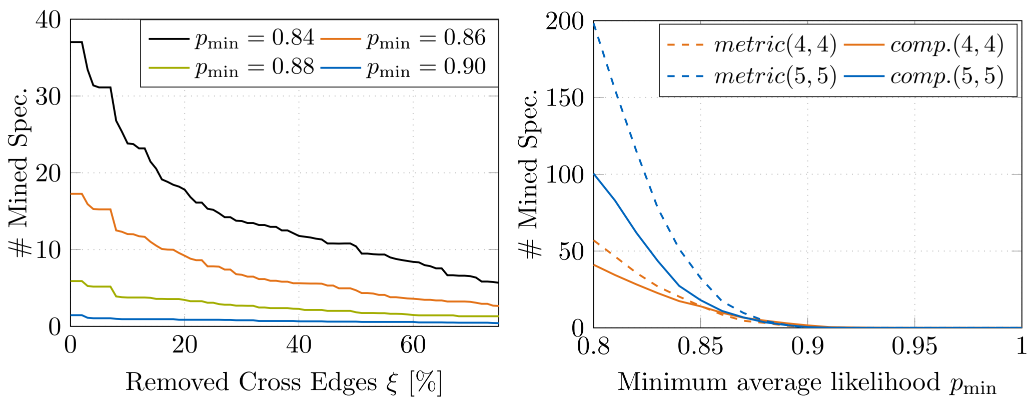


Figure 8.6: *Left:* Number of mined specifications under various percentages of removed cross-edges ξ between original and validation BN, under various minimum average likelihoods $p_{\min}$. *Right:* Comparison of metric based and comparison based approach in terms of # Mined Specifications for various minimum average likelihoods $p_{\min}$ for 2 different BN sizes [10].

contain events from more than one functional process (i.e. from RVs of different BNs). Lastly, the *run time* of the approaches is measured.

8.4.1.2 Data Set

In the evaluation of [10] synthetic MSSs are generated with the same generator as the one introduced in Chapter 7. This is done by sampling from multiple TSCBNs that each represent one functional procedure that is to be modeled. Random MSSs that are sampled from each TSCBN are shifted randomly in time and concatenate forming a trace of events of the shape (time, RV, value). To be able to distinguish between functional procedures in the result set and for easier measuring of false positives RVs per TSCBN are unique across all TSCBNs. To simulate dominant behavior of functions, the CPD of a randomly picked state per RV is set between 0.6 and 1.0. The generated TSCBNs have between 4 and 5 TVs, 4 to 5 nodes per TV, 2 states and $0.8 \cdot \#$nodes inter-edges to 3 other TVs per TV.
The minimum literal ratio and maximum combination count is set to 0.5 and $2.5^{n/2}$ respectively, where n is the *regex*' number of symbols.

8.4.1.3 Results

In [10] three experiments were presented, that underline the expressiveness and good false positive rate of the approach. This was shown in the following three cases.

Evaluating structural variation in the comparison based approach: Strictness of specification depends on the similarity of the models in the comparison based approach and $p_{\min}$ can be used as a parameter to filter for expressive specifications.

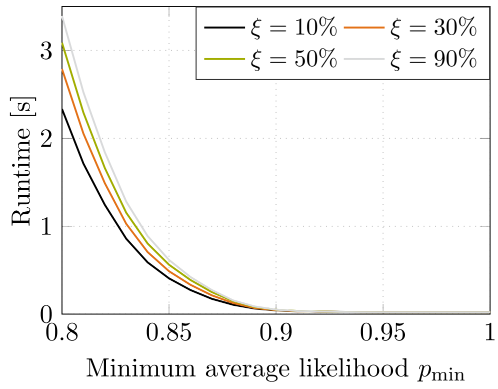


Figure 8.7: Run time of the approach under various average likelihoods $p_{\min}$ for 4 cross-edge deletion percentages ξ [10].

There, Figure 8.6 were produced by running the comparison based approach on 50 iterations produced by one TSCBN and its validation TSCBN. Here, in Figure 8.6 (left) it was shown how the quality of the resulting specification varies when the number of cross-edges is modified. There the validation BN's structure was modified by randomly removing 0 to 75 % of cross-edges from the original BN for several minimum average weight thresholds $p_{\min}$.
The results show that less specifications are found if more edges are removed. This is, as with less overlapping edges more potential specifications violate the validation model and thus, more candidate specifications are dismissed. Further, the merging blurs the candidate specification until the threshold strictness metric is violated. This shows that stricter specifications are found from more similar TSCBNs without merging those potentially too often based on the metrics.
Further, in Figure 8.6 (right) it is seen that curves with higher $p_{\min}$ produce less specifications. This is as higher values require more expressive specifications to be mined only while unlikely ones are dropped. With this, $p_{\min}$ allows to parameterize the level of strictness of BaySpec.

Evaluating specifications for both approaches: The metric based approach, is less strict than the comparison based approach, with this finding more potentially less expressive specifications. For this, both approaches were tested for 2 model sizes with (TV, nodes per TV) of (4,4) and (5,5), each sampled 50 times randomly. Also, in both approaches literal ratio is set to the range [0.5, 0.8] and the combination count to $[2^{n/5}, 2^{n/2}]$, where n is the *regex*' number of symbols. The experiment shows, as can be seen in Figure 8.8, that the metric based approach finds more specifications as it is only limited by its metrics, while additional restriction of the structure of the validation model (here 20% removed cross-edges) yield less but more meaningful specifications. Also, in Figure 8.9 it was shown that for both approaches $p_{\min}$ defines the meaningfulness of the found specifications, with too low values yielding many but meaningless expressions, while too high values yield the opposite.

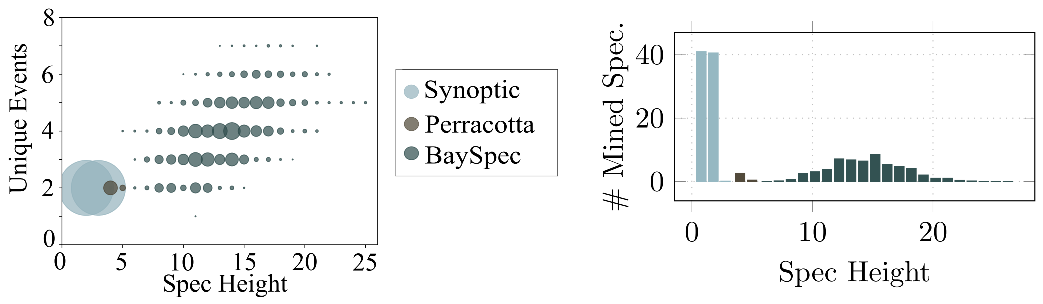


Figure 8.8: *Left:* Height and number of unique events of found specifications for three approaches, with circle size being frequency of occurrence. *Right:* Height of Specifications against its frequency [10].

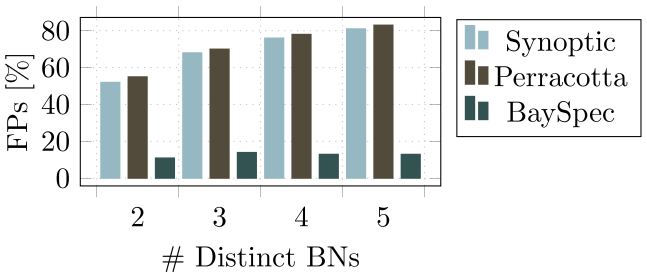


Figure 8.9: Ratio of FPs mined by three approaches with traces of increasing number of functions [10].

Evaluating runtime of approaches: Both approaches yield a reasonable runtime and are thus, well applicable to be used on TSCBNs in the proposed DM pipeline. This was demonstrated by measuring the runtime of the comparison based approach and all its steps, which include Mining Graph extraction and path computation, measured and averaged over 50 models of above structure, where $p_{\min}$ is increased at various removed cross-edge ratios ξ. It is found and shown in Figure 8.7, that higher run times are resulting for lower values of $p_{\min}$. This is as such values produce more paths that require more merging and model checking operations to be performed.

Evaluation against existing approaches: It was also shown, that comparable approaches such as Perracotta [71] and Synoptic [64] were outperformed in terms of expressiveness and false positive rate. This was done as follows.
To make it comparable to BaySpec, Synoptic is extended such that it is able to find specifications with a confidence below 1, with this being able to handle imperfect traces. Complexity was compared by randomly sampling from TSCBNs to get trace sets of various complexities, i.e. the number of traces is varied per trace set and the number of sampling repetitions within one trace, both between 2 and 5. These trace sets are used as inputs for all three the tools. In the validation BNs, 20 % of cross-edges are removed and $p_{\min}$ is set to 0.85. Resulting complexities are illustrated in Figure 8.8 (*left* and *mid*), with bigger circles indicating more frequent occurrence of complexities. As it is shown *BaySpec*'s extraction produces more variable arbitrary length patterns of higher complexity, which results from capturing longer sequences of event correlations.
In a second experiment, shown in Figure 8.9 the false positive rate was evaluated. Here each functional procedure is represented by one TSCBN and a pool of 50 TSCBNs, each representing one functional procedure, is randomly generated. Then, an increasing number of TSCBNs were used to produce trace sets of multiple functional procedures, that occurred multiple times. On the resulting trace, Synoptic and Perracotta were run and the FP ratio measured. Further, it is assumed that *BaySpec* learned the ground truth models and a validation model. That is, to simulate structural loss during training, in the validation model 30% of cross-edges are removed and $p_{\min}$ is set to 0.8. On those two models BaySpec is run. To now estimate FPs from BaySpec the following was assumed. In *BaySpec*, mined specifications might match paths that were not used during merging. That is, during merging *regex*s are found that match paths that did not contribute to the merging process. Thus, for BaySpec properties with a combination count bigger than the number of merged paths are considered as FPs, i.e. the number of such possible paths that did not contribute to the merging process. This is the worst case estimate. It was shown that the other approaches tend to mix functional behaviors and, thus, produce high numbers of FPs. However, if prior segmentation is used that generates one TSCBN per functional procedure this results in specifications with less FPs. This shows BaySpec to be well suited to be used as the last step of the proposed DM pipeline.

8.4.2 Case Study

Based on [10], in this section it is demonstrated how the TSCBNs are used for the effective extraction of specifications from MSSs. For this two real world datasets are analyzed that were recorded from a fleet of test vehicles of a big automobile company.

Data set	# Samples	# Temp. Variables	# States / # Nodes	latent sequences	mean log-likelihood	temporal log-likelihood
High Beam	80	5	2.00	0.0 %	-0.18	-19.59
Indicator	104	5	3.67	28.6 %	-0.23	-44.78

Table 8.3: Properties of the automotive datasets including results after model creation and parameter estimation [10].

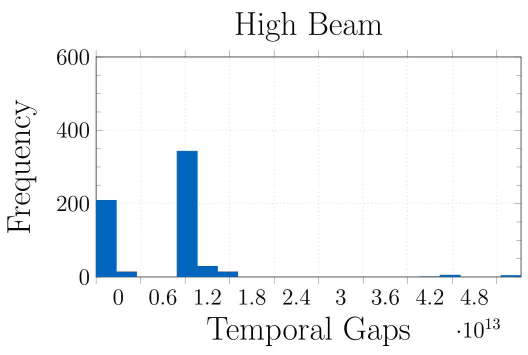
This is done with TrieDiscover and EM Estimation to learn the TSCBN followed by BaySpec and MPE estimation for Specification Extraction. Both Case studies were executed in the context of the large scale distributed system which is the in-vehicle network in a car, similarly as presented in Chapter 5. That is MSSs of different functional procedures of a car are produced by the system and analyzed here.

8.4.2.1 Data Sets

Two data sets were considered.

High Beam Assistant: The functional procedure of the high beam assistant is defined as follows. On approaching traffic the high beam lights are automatically switched off, while those are switched on if no traffic is present. But only, if the assistant is active, else this functional procedure is not observed. The according MSS contains five TVs. Those include the light state S_{staHB}, which defines if the light is on or off. The lights control signal S_{ctrHB} turns the light *on* or *off* if enforced by the system. Further, the control of the assistant has the following TVs. That is, the control button S_{ctrAS} which can turn the assistant *on* or *off*, as well as the current state of the assistant S_{staAS} which can also be *on* or *off*. Lastly, the TV S_{det} defines if approaching traffic was found, i.e. it can take on the states *detected* if traffic was detected and *not detected* if this is not the case.
With this, if the assistant is *off* the beam state is not changed on approaching cars whose detection is indicated as *detected* in S_{det}. However, if the assistant is turned *on*, each detection causes S_{ctrHB} to turn *on* S_{staHB}. Multiple MSSs of activations and deactivations were observed of the high beam and of the assistant.

Indicator: The functional procedure of the indicator is described in [8] in the following way. The indicator light is switched on by a driver using the handle bar, when turning left or right. This indicator function is comprised of five TVs. It consists of a handle S_{bar} that *(de-)activates* the indicator either in *steady mode* or for *3 seconds*. Which of those two types was chosen is represented in the TV S_{type}. Depending on the state of the handle bar the indicator S_{state} changes its state, e.g. if it was turned on it might yield *right indicator on*. If this activation was performed in steady mode, synchronization S_{sync} is *started* and a *new indication cycle* S_{cycle} begins. Once S_{bar1} changes back to its *default state* from the *3 second* state, depending on this state synchronization is *stopped*. If it is in *steady mode* it is only *stopped* at S_{bar2}, when being returned to *default* by the user.



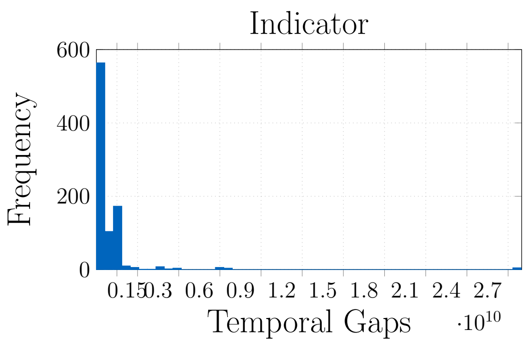


Figure 8.10: Distribution of gaps between consecutive state changes in nanoseconds for the high beam and indicator data set [10].

This was done for two different cars. In the first case, 223 MSSs were extracted, where S_{sync} was omitted for a more comprehensive representation of the results. In the latter case, all TVs and 104 MSSs are considered giving the results of [8]. Further in [8] properties of the data set are provided and given in Table 8.3. Two structures are learned (each from an equal split of the data set) with TrieDiscover and EM, where a pessimistic and naive choice on edges was chosen resulting in more inter-edges. In [8] the choice of parameters was evaluated in more detail, which resulted in a more suitable choice of parameters and thus, in a reduced representation of the extracted TSCBN. For this case, the learning and parameter estimation of the network is discussed in more detail. On all three data sets BaySpec was applied and on the two data sets of [8] also MPE is applied.

8.4.2.2 Experiment

In [8] the process of finding appropriate parameters for TrieDiscover is described. This process was used to determine good parameters for the results of [8] and is revised in the following.

Metrics: As defined in [8], the found LTL specifications are evaluated in terms of complexity. Also in the case of [8] an expert manually labeled found specifications as either fully right or only partly right. The latter implies that at least one literal is placed badly, such that malicious firing of the specification might occur. The accuracy resembles the ratio of specifications that are fully right. Further, from the accurate traces the found specifications are labeled as either trivial, i.e. specifications that always fire, or relevant, i.e. specifications that fire in a critical case which might reveal an anomalous spot in the trace.

Parameterization of Model: In this case TrieDiscover was used with PC structure optimization which has parameters t_{th} , k and χ_{th}. Those were found as follows.

- **Choosing t_{th}:** t_{th} defines the temporal threshold which indicates how far back each state change in the TSCBN can exhibit correlation from another state change. By inspecting the distribution of gaps between consecutive state changes in the data sets an estimate for this value van be found. For the two data sets of [8] this distribution is shown in Figure 8.10. t_{th} of TrieDiscover is chosen based on this as follows. To avoid ignorance of potentially relevant temporal dependencies a pessimistic estimate for t_{th} is to include the last two preceding state changes (i.e. twice the highest gap size). That is, for the high beam data set $t_{th} = 2 \cdot 2 \cdot 10^{13}$ and for the indicator data set $t_{th} = 2 \cdot 4 \cdot 10^{9}$.

- **Choosing k and χ_{th}:** With the found t_{th} TrieDiscover is run with different values of k and χ_{th} and the number of inter-edges is measured. If a reasonably high correlation between state changes is found, a higher number of inter-edges is preferred over a low number as more information is included in that case. To thus, evaluate the number of inter-edges for those parameters, the approach was run for values of k between 0.05 and 1.00 and for χ_{th} between 0.05 and 100.00 resulting in inter-edges as shown in Figure 8.11. For the high beam this gave $k = 0.05$ and $\chi_{th} = 5.00$, while for the indicator this gave $k = 0.2$ and $\chi_{th} = 0.25$.

With this, the structure of Figure 8.13 was found for the data sets in [8]. This structure was shown to be meaningful according to the discussion in [8]. For the data set of [10] the resulting structure is shown in Figure 8.12. With those structures, for all data sets parameter estimation is performed using the EM Algorithm with 5 iterations. Also, overfitting is a desired property in this scenario, as the approaches do not aim for generalization but rather for model fitness.

Parameterization of BaySpec: $p_{\min} = 0.8$ was chosen and BaySpec was run with the comparison based approach on the learned TSCBN. In [8] BaySpec was run on its TSCBN with the metric-based approach that used $p_{\min} = 0.6$ as value. Here a smaller value was chosen as less inter-edges are present, which gives a bigger result set for merging.

Parameterization of MPE The 500 most likely MPE estimates in state and time are found sampling 100 000 times from the trained TSCBNs. Most likely drawn combinations are then, ranked according to its occurrence frequencies and assigned a time as described in Section 8.2.

Execution of Approaches: BaySpec produced specifications of complexities shown in Figure 8.14. Excerpts of the most likely discovered LTL specifications of [8] are shown in Table 8.4. Here, metrics of the found specifications are provided in Table 8.5. The MPE estimates found in [8] are revised in the results section here.

8.4.2.3 Results

BaySpec: As can be seen in Figure 8.14 similar to the synthetic case, multiple specifications were found for all three approaches. However, in the case of BaySpec again found specifications are expressive and of various length as circles are on the top right. In contrast to that, Perracotta and Synoptic produce lower complexity specifications. This shows that even in a noisy real world scenario BaySpec is well suited to produce

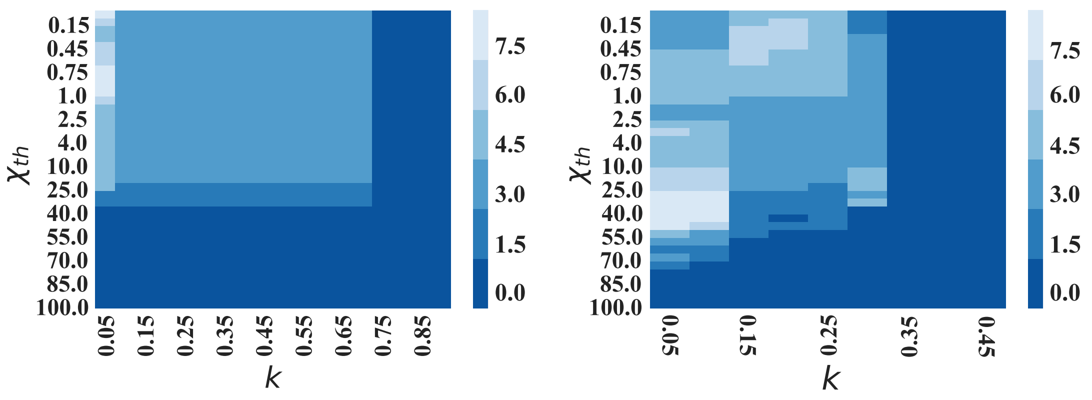


Figure 8.11: Number of inter-edges after structure discovery for different values of k and χ_{th}, with the high beam shown on the left and the indicator results shown on the right [10].

expressive specifications with the comparison based approach.

BaySpec Indicator [8]: A more detailed inspection of found real-world specifications was given in [8]. There the LTL specifications are discussed with the list given in Table 8.4. BaySpec found 28 specification from the indicator TSCBN, There, the first LTL formula suggests that if the handle bar was tipped up and the synchronization started a new cycle, the synchronization has to continue, before then, a new cycle is started. This was a dominating MSS in the training set. The second formula implies that a movement of the handle bar needs to return to its default state, while the last formula says that the synchronization sequence that is described in the first formula is started once the handle bar was triggered by the user.

BaySpec High Beam [8]: BaySpec found 17 specification from the high beam TSCBN. The first formula in Table 8.4 implies that if the high beam was turned on before and a detection is noted the control will turn the high beam off which consequently turns the stat off the high beam off. In the second formula , if the high beam is controlled on it has to change its state to be on or it has to eventually turn off. In the third formula, similar to the first case, if detection was noted the control has to turn eventually off the high beam.

According to Table 8.5 it was further shown that a good accuracy is reached for both data sets. The discrepancy to 100 % was explained in [8] with an imbalance in both data sets, as follows. That is, in the indicator data sets, as users tend to use the short indication far more often then, the permanent indication. Therefore, the non permanent case is well represented while the permanent case yields CPDs that are mostly approximated during the EM approach. Similarly in the high beam assistant the relation between the assistant being on or off and the high beam activating automatically is not represented in the TSCBN. This, can be adjusted in future applications by either including balancing or including expert input at this stage. The complexity and expressiveness of the found

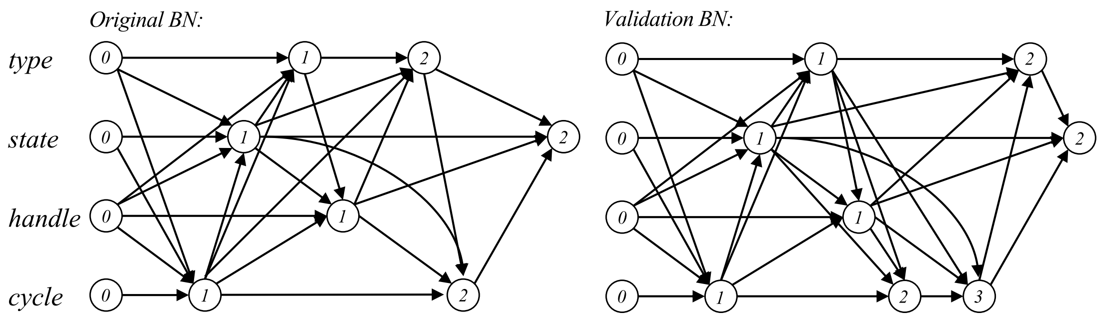


Figure 8.12: Original and validation BN for the indicator activation function [10].

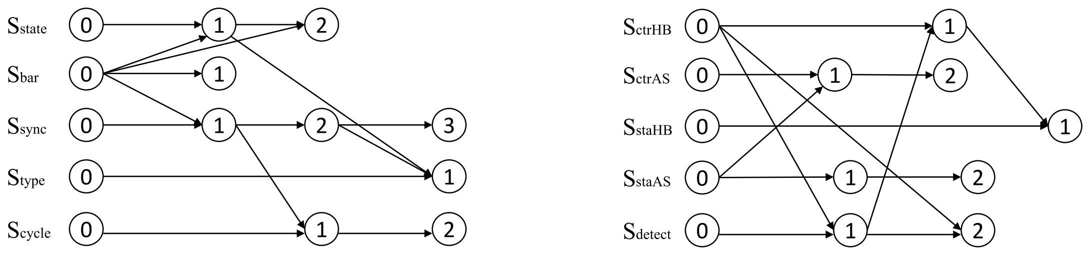


Figure 8.13: This figure shows the network that was discovered with TrieDiscover for the indicator on the left and for the high beam assistant on the right [10].

specifications yields a good quality, with a high percentage of relevant specifications and specifications of heights around 5 and 3 to 4 symbols included per specification. This shows, that TSCBNs can be used to automatically learn LTL specifications from observed system behavior if a TSCBN is given. With this, it is well suited as the last step of the proposed DM pipeline.

MPE High Beam [8]: After sampling in the high beam TSCBN the most frequent MPE estimates appeared in 5.5 % cases and the least frequent in 0.009 %, where the two most likely estimates of the high beam are with 5.5 % frequency

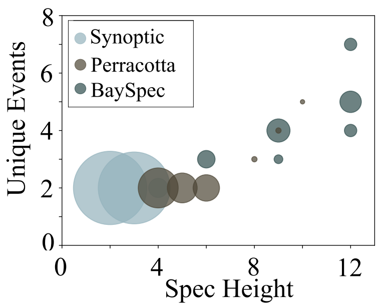


Figure 8.14: Height and number of unique events of found specifications for three approaches, with circle size being frequency of occurrence [10].

$\mathcal{L}$ [%]	Discovered LTL - Specifications
87.37	$\mathbf{G}((S_{bar}\text{: tip up} \vee S_{sync}\text{:cyc. continues}) \mapsto \mathbf{XG}((S_{sync}\text{:new cyc.} \mapsto \mathbf{X}((S_{sync}\text{:sync} \wedge \mathbf{X}(S_{sync}\text{:cyc. continues}))))))$
86.90	$\mathbf{G}((S_{bar}\text{:overtip down} \vee S_{bar}\text{:tip up}) \mapsto \mathbf{X}(\mathbf{F}(S_{bar}\text{:no action})))$
86.36	$\mathbf{G}((S_{bar}\text{:tip down} \vee S_{bar}\text{:tip up}) \mapsto \mathbf{X}(\mathbf{F}((S_{sync}\text{:new cyc.} \wedge \mathbf{X}(S_{sync}\text{:sync} \wedge \mathrm{X}(S_{sync}\text{:cyc. continues}))))))$
84.57	$\mathbf{G}(S_{ctrHB}\text{:on} \mapsto \mathbf{X}(\mathbf{F}(S_{detect}\text{:detection } \mathbf{U} \ (S_{ctrHB}\text{:off} \wedge \mathbf{X}(\mathbf{F}(S_{staHB}\text{:off}))))))$
82.91	$\mathbf{G}(S_{ctrHB}\text{:on} \mapsto \mathbf{X}(\mathbf{F}((S_{staHB}\text{:on} \vee S_{ctrHB}\text{:off}))))$
73.28	$\mathbf{G}((S_{ctrHB}\text{:off} \vee S_{detect}\text{:detection}) \mapsto \mathbf{XG}(S_{detect}\text{:detection} \mapsto \mathbf{X}(S_{ctrHB}\text{:off} \wedge \mathbf{X}(\mathbf{F}(S_{staHB}\text{:off})))))$

Table 8.4: Excerpt of LTL Specification found with BaySpec. The upper part shows the indicator and the lower half part shows the high beam results, with likelihoods of found specifications $\mathcal{L}$.

dataset	# LTL Spec.	$\varnothing$*height*	$\varnothing$*width*	accuracy [%]	relevant [%]
High beam	17	5.29	3.35	76.47	61.05
Indicator	28	5.75	4.03	67.85	84.22

Table 8.5: Metrics for the specifications that were extracted with BaySpec.

- $S_{ctrHB} = (on, \Delta t = 0.0s), (off, \Delta t = 0.743s)$
- $S_{ctrAS} = (no\ action, \Delta t = 0.0s)$, $(no\ action, \Delta t = 0.0s), (no\ action, \Delta t = 0.0s)$
- $S_{staHB} = (off, \Delta t = 0.0s), (off, \Delta t = 0.006s)$
- $S_{staAS} = (on, \Delta t = 0.0s), (on, \Delta t = 0.017s), (on, \Delta t = 1.682s)$
- $S_{detect} = (detection, \Delta t = 0.0s), (detection, \Delta t = 0.0s), (detection, \Delta t = 0.0s)$

and with 4.3 % frequency

- $S_{ctrHB} = (off, \Delta t = 0.0s), (on, \Delta t = 0.836s)$
- $S_{ctrAS} = (no\ action, \Delta t = 0.0s), (no\ action, \Delta t = 0.0s), (action, \Delta t = 0.0s)$
- $S_{staHB} = (off, \Delta t = 0.0s)$, $(on, \Delta t = 0.048s)$
- $S_{staAS} = (on, \Delta t = 0.0s)$, $(on, \Delta t = 0.017s), (on, \Delta t = 1.682s)$
- $S_{detect} = (no\ detection, \Delta t = 0.0s), (no\ detection, \Delta t = 0.0s), (no\ detection, \Delta t = 0.0s)$

As described in [8], it can be well seen that the upper estimate capture the actions of both the high beam controller and the assistant. More precisely, the detection has triggered to change S_{ctrHB} from on to off, and the variable S_{staHB} when already off to remain off. Moreover, the assistant performed an action at the last step of his process. Notably, although disjoint the dominant state further tells that the assistant was on when the detection activated the high beam.Above that, temporal gaps Δt for TVs without state change are not meaningful as the TVs do not change its state. The second estimate shows in state and time the inverse case where the non-detection of traffic caused the high beam to change from off to on, which in turn changed the state from off to on, while the assistant process is identical to the preceding case. This shows how TSCBNs capture interval and state change relations of a process in an interpretable manner.

MPE Estimations - Indicator: For the indicator frequencies are within 11.99 % and 0.017%. The two most likely estimates are with 11.99 % frequency

- $S_{bar} = (tip\ up, \Delta t = 0.0s)$, $(no\ action, \Delta t = 0.513s)$
- $S_{sync} = (cyc.\ cont., \Delta t = 0.0s)$, $(new\ cyc., \Delta t = 0.017s)$, $(sync, \Delta t = 0.639s)$, $(cyc.\ cont., \Delta t = 2.052s)$
- $S_{type} = (non\ permanent, \Delta t = 0.0s)$, $(non\ permanent, \Delta t = 0.0s)$
- $S_{state} = (both\ off, \Delta t = 0.0s)$, $(right\ on, \Delta t = 0.017s)$, $(both\ off, \Delta t = 1.682s)$
- $S_{cycle} = (no\ ind., \Delta t = 0.0s)$, $(normal, \Delta t = 0.0s)$, $(no\ ind., \Delta t = 2.660s)$

and with 4.4 % frequency

- $S_{bar} = (tip\ down, \Delta t = 0.0)$, $(no\ action, \Delta t = 0.572s)$
- $S_{sync} = (cyc.\ cont., \Delta t = 0.0)$, $(new\ cyc., \Delta t = 0.020s)$, $(sync, \Delta t = 0.639s)$, $(cyc.\ cont., \Delta t = 2.052s)$
- $S_{type} = (non\ permanent, \Delta t = 0.0s)$, $(non\ permanent, \Delta t = 0.0s)$
- $S_{state} = (both\ off, \Delta t = 0.0s)$, $(left\ on, \Delta t = 0.0204s)$, $(both\ off, \Delta t = 1.675s)$
- $S_{cycle} = (no\ ind., \Delta t = 0.0s)$, $(normal, \Delta t = 0.181s)$, $(no\ ind., \Delta t = 2.660s)$

For this case the indicator learned both the activation of the right indicator after the handle bar was tipped up and the left activation when tipping down. Both caused a new indication cycle to start. Notably, here the timing was learned nearly exact, e.g. it is plausible that after tipping up the indicator starts 0.017 seconds later and stops indication after around 1.7 seconds (S_{state} *2*) and confirms this at around 2.6 seconds in S_{cycle} *2* with the no indication signal. In practice, this approach allows an expert to first, directly understand the prevalent system behaviors (e.g. for fault diagnosis) and to second, deduce potential specifications in state and time in a fully automated manner. This is, as snapshots of functional procedure variants of MSSs are compactly represented as outcomes of learned TSCBNs.

8.5 Summary and Conclusion

In this chapter two approaches for the extraction of specifications and dominant behavior are described (as first introduced in [10, 8]). Given a TSCBN that represents behavior of observed MSSs both approaches assume that most likely constellations resemble specified behavior of functional procedures in a trace. Assuming that unlikely behavior is not observed significantly often, noise is handled here.
BaySpec extracts LTL specifications by identifying most probable paths and merging those until specifications of appropriate strictness are found. The resulting specifications are assumed to represent specifications that are relevant to the analyzing domain, as both segmentation and TV clustering was already performed in a supervised manner. A major limitation of BaySpec is that it does not include temporal information and

resulting specifications do not directly allow to understand the dominant behavior that is present in the set of MSSs. This is solved with MPE. By sampling from the TSCBN MPE allows to identify most probable constellations of the MSSs that were observed in state and time. Based on this experts are able to better understand the prevalent behaviors of the procedure. This in turn, can be used as a basis for generation of specifications either by using the dominant state model as a template or by extending the expert's knowledge.

9 Case-Study: Specification Mining in Automotive In-Vehicle Network Traces

In the previous chapters all steps of the DM pipeline were evaluated individually. Next, in this chapter the pipeline is evaluated all together. That is, its consistency and effectiveness is demonstrated on several real world data sets from the automotive industry. The pipeline consists of multiple stages each of which might use different approaches, has multiple parameters and is evaluated in terms of different metrics. Therefore, in Section 9.1 an overview on the configuration that is used in this implementation is given. Next, the motivation and background in the automotive domain is introduced in Section 9.2. Five data sets, that each contain multiple functional procedures of a certain domain, were used in this evaluation and are described in Section 9.3. Based on the implementation the results of the specification extraction procedure are presented and discussed in Section 9.4.

9.1 Implementation

At each step multiple approaches can be applied, while not all are equally well suited for the task of Specification Mining. Based on the evaluations given in the previous chapters of each stage, a defined implementation is used at each stage, which is introduced in this section and shown in Figures 9.1 and 9.2. Based on this overview, a more detailed discussion of each step is presented in the respective section of this chapter.

1. **Preprocessing:**
   - *Goal:* Given a raw trace, as introduced in Chapter 4, an interpreted trace that contains a subset of relevant TVs is extracted. For this, the expert defines a coarse subset of relevant TVs which is related to his domain.
   - *Input / Output:* A raw trace $\mathbf{K}_b$ is input. Based on a set of selected TVs $\mathbf{U}_{rel}$ a reduced trace $\mathbf{K}_{cond}$ is extracted, which is processed per type, which yields the MES $\mathbf{K}_s$.
   - *Approach:* The approach of Chapter 4 is used here, where *Preselection*, *Interpretation*, *TV splitting* and *Constraint-based reduction* is done on a cluster using Apache Spark. During type-dependent processing numerical TVs are discretized using SAX and ordinal TVs are treated as nominal. Further outliers are removed automatically as described in Chapter 4 and manually by flagging irrelevant values of TVs. Also, TVs that do not change its value at least ones are dropped.
   - *Parameters:* Preprocessing has parameters $\mathbf{U}_{rel}$, C and E. In this evaluation we do not extract extensions and thus, $E = \emptyset$. Per data set that is

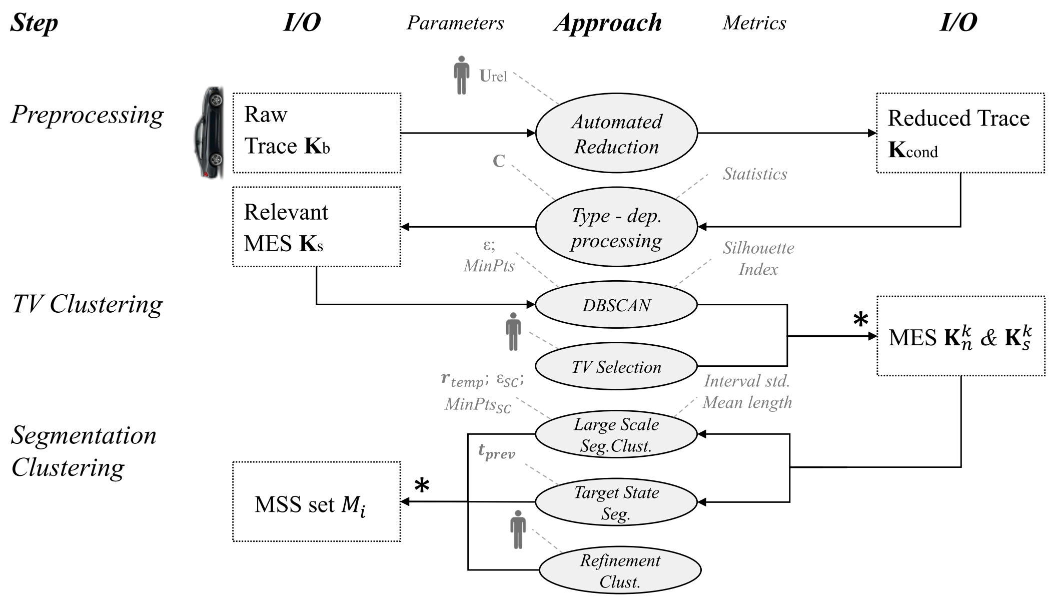


Figure 9.1: First part of the overview of the configuration used for the evaluation in this chapter.

investigated a list of TVs, that correspond to a domains function, is given in $\mathbf{U}_{rel}$. C is defined such that only changes of TV values are kept.

- *Section:* The results for this stage are presented in Section 9.4.1.

2. **TV Clustering:**
   - *Goal:* Per data set groups of TVs are identified.
   - *Input / Output:* The MES $\mathbf{K}_s$ and expert knowledge is input here. The results of this step are multiple relevant groups of TVs that correspond to common functions. The resulting groups yield a MES $\mathbf{K}_s^k$ and a MSS $\mathbf{K}_n^k$ of TVs which is taken as a basis for selection of TVs that are used for further processing.
   - *Approach:* The approach in Chapter 5 is used with DBSCAN, which is well suited to find groups of TVs as was shown there. Based on the found clustering, experts choose TVs that are of relevance for further inspection.
   - *Parameters:* Parameters include the window size and DBSCAN's ϵ and $MinPts$. The window size is set fixed to 400 overlapping windows, while optimal values for ϵ and $MinPts$ are found using a grid search.
   - *Metrics:* To evaluate clustering performance and to find good hyper parameters the Silhouette index was chosen as a metric.
   - *Section:* The results of hyper parameter estimation and the insights of this stage are presented in Section 9.4.2.

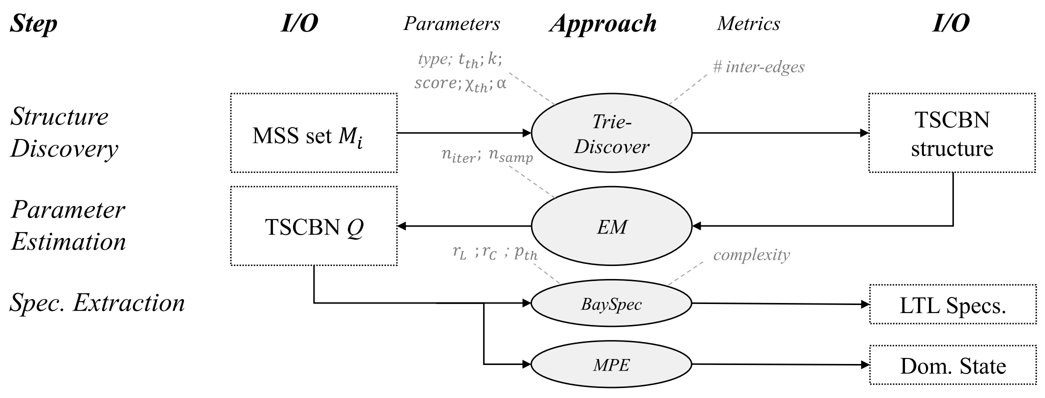


Figure 9.2: Second part of the overview of the configuration used for the evaluation in this chapter.

3. **Segmentation:**
   - *Goal:* Each sequence of TVs contains multiple functional procedures. In order to extract those, segmentation and clustering is used in the way it was introduced in Chapter 6.
   - *Input / Output:* The input at this step is an MSS $\mathbf{K}_n^k$, which was chosen in the previous step, as well as expert knowledge to interactively refine the clusterings. This results in multiple sets M_i of MSSs, that each correspond to one functional procedure.
   - *Approach:* Here, two approaches are used to find M_i. First, this is the approach presented in Chapter 6 and Section 6.4, which showed to scale best while yielding solid accuracy. Here, DBSCAN is used for clustering. Second, for some data sets a target state is known and the behavior before that target state is investigated. In this case a simple segmentation is used, where a window t_{prev} before the target state change is used to extract one set of MSSs M_i. Given those clusterings the expert might visually sub clustering those in a hierarchical manner.
   - *Parameters:* For the further approach the temporal range r_{temp} around each data point is used. Also, the DBSCAN clustering of segments has parameters ϵ_{SC} and $MinPts_{SC}$. For the latter approach the window size t_{prev} defines segmentation quality.
     It is assumed, that optimal MSS sets M_i have a similar number of state changes per TV, with temporal gaps of similar lengths. Thus, optimal parameters that optimize this are found by using a grid search on below metrics.
   - *Metrics:* Two metrics are used here. First, that is the mean of the standard deviations of the number of state changes per TV. If this standard deviation is low a similar number of TVs is given. Second, as optimization of the standard deviation might prefer short (i.e. non-informative) sequences, as a second metric the mean number of state changes per TV is used here as well.

- *Section:* The results of hyper parameter estimation and the insights of this stage are presented in Section 9.4.3.

4. **Structure Discovery:**
    - *Goal:* Each MSS set M_i represents a functional procedure. At this step the structure of a TSCBN is learned from M_i to represent the procedure under uncertainty.
    - *Input / Output:* The input of this step is a MSS set M_i and the parameters for TrieDiscover. Those are the type *type*, a temporal threshold t_{th} the filtering parameter k, score type *score*, as well as thresholds α and χ_{th}. The output is the structure of the TSCBN.
    - *Approach:* TrieDiscover, which was introduced in Chapter 7 is used to learn the structure. All three variants of TrieDiscover are considered here during evaluation.
    - *Parameters:* Those depend on the variant of TrieDiscover, which is indicated as *type* in Figure 9.2. A parameter that is required in all variants is the temporal threshold t_{th} and the filtering parameter k. sbTD further uses the score type *score* which is chosen to be BIC here. Also, cbTD and cbvTD both have α as parameter and cbvTD has χ_{th}.
    An optimal structure contains all important inter-edges as well as a representative number of nodes per TV. For Specification Mining an optimistic choice is preferable, i.e. it is better to represent too many inter-edges than choosing to less. That is, as parameter estimation will make CPDs that correspond to dominant observed sequences (rather than approximated parts of those) more likely. Also, even if a path was observed less often and thus, the correlation for the corresponding sequence can not be tested with high confidence, this path might still represent a specification. In particular, paths that are not present can not be used for specification extraction, while notably, the structure learned by TrieDiscover is loss less. That is, the candidate set of parents does only result in inter-edges that are plausible according to the data. Thus, using too many edges always results in logical paths that are likely to correspond to at least one observation.
    t_{th} is found by inspecting the distribution of gaps between consecutive state changes as done in the case study of Chapter 8. α, χ_{th} and k are found by using grid search.
    - *Metrics:* The metric used for optimization is the number of inter-edges. As argued above a high number of inter-edges is preferable.
    - *Section:* The results of hyper parameter estimation and the insights of this stage are presented in Section 9.4.4.

5. **Parameter Estimation:**
    - *Goal:* Given the structure parameters of the TSCBN are learned from M_i.
    - *Input / Output:* The input is the structure of the TSCBN, the estimation approach with its parameters and the set M_i. The output is a trained TSCBN Q that represents the dynamic behavior of the MSS set.

- *Approach:* As was shown in Chapter 7, the three approaches MLE-R, VI and EM gave comparable results and thus, are equally suited to be used here. In this evaluation EM and MPE-R are used.
- *Parameters:* EM has mainly two parameters which is the number of samples n_{samp} drawn for estimation of expectations and the number of iterations until convergence n_{iter}. To keep computational complexity low $n_{samp} = 1000$ and $n_{iter} = 5$ were chosen.

6. **Specification Extraction:**
   - *Goal:* Based on the learned TSCBN the most likely behavior is extracted and processed in order to determine specifications or dominant behavior of the system.
   - *Input / Output:* The input of this step are a learned TSCBN Q that represents a functional procedure of the system, as well as the parameters for the approaches. The output is a list of LTL specifications when running BaySpec and a list of most likely system states when performing MPE.
   - *Approach:* For Specification Mining the metric-based approach of BaySpec and the MPE approach of Chapter 8 are used.
   - *Parameters:* BaySpec has the parameters r_L, r_C and p_{th}. Optimal parameters produce specifications of higher complexity and high likelihood, as it is assumed that this corresponds to more informative content.
   - *Metrics:* The height and width of specifications, which were presented in Chapter 8, indicate the complexity and thus, the information content of the specifications. Notably, BaySpec produces specifications of appropriate strictness as this is limited by r_L and r_C.
     The plausibility of specifications is hard to be represented in terms of metrics. Thus, it is assessed manually based on an expert.
   - *Section:* The results of hyper parameter estimation and the insights of this stage are presented in Section 9.4.5.

9.2 Background: Automotive Verification

In-vehicle networks of a car are an example of large-scale distributed system, as those run distributed software across multiple Electronic Control Units (ECU). Their functionality is implemented by multiple domains in terms of hardware, communication and software, before being integrated into the overall system of a car. To verify functionality within such vehicles data-driven testing is used. For this, multiple cars are taken on test journeys, where a set of test cases are applied on those systems to ensure that multiple functionalities were covered by execution. Those executions are recorded as traces that are used for both verification with existing specifications and for mining of those.
In 2015 a modern vehicle ran over 100 million lines of source code on-board, with up to 15 ECUs communicating per function, while transmitting 2 million messages per minute. Resulting traces are massive in size (e.g. at BMW Group 500 cars produce 1.5 TB per

day) and thus, require automated methods for verification. In particular manual generation of specifications on the integrated system is in-feasible, which is why automated methods such as the one proposed in this thesis could be used in the future to ensure meeting the high quality requirements of such systems.
The recorded execution logs are in raw format and are comprised of messages that are transmitted between ECUs. Data encoded in such messages is highly redundant and massive in size. Thus, direct application of Specification Mining approaches is not possible. However, by applying the overall DM pipeline of this work, this is made possible and allows to find specifications of the vehicles' system states.

9.3 Data Sets

Five data sets are extracted from journeys of different cars. For each of those data sets an expert defined a coarse subset of relevant TVs, that categorizes those for evaluation. Notably, those sets only capture a subset of the behavior of those functions. This is due to the fact that only system states that are sent on the bus are observed. However, this suffices for the verification of this methodology. The following data sets are used.

Active Cruise Control (ACC): This data set contains TVs that are relevant to the activation and deactivation of the ACC functionality. It contains a set of manually preselected TVs that are known to affect those target state changes. In this data set the aim of the case study is to identify which state combinations and state changes resulted in the target changes of activation or deactivation.

Lights: This set contains TVs that affect the light functionality. It contains state changes that relate to the functioning of the lights, which among others includes front lights, blinker and high beam transitions that are manually or automatically changed. Here functional procedures are extracted with the overall approach. This data set is especially well suited, as it contains multiple functional procedures that relate clearly to different system behaviors.

Start up Procedure: In this data set a preselected set of TVs is contained that relate to both the starting and the ending of the car, such as engine behavior. Here functional procedures are extracted with the overall approach.

Shut Down Procedure: In this data set another set of TVs, that is related to both the starting and the ending of the car, is contained. However, here the set of TVs is chosen more coarsely. Further, this set was already pre-processed by experts such that discretizations of some numerical TVs was performed already based on suitable thresholds. Here again functional procedures, specifications and dominant behavior is extracted with the overall approach. Naming of the latter two data sets was chosen such that those are clearly distinguished.

Wiper: This data set has a set of preselected TVs that relate to the wiper in the car. This includes manual and automatic activations and deactivations of it, as well as its

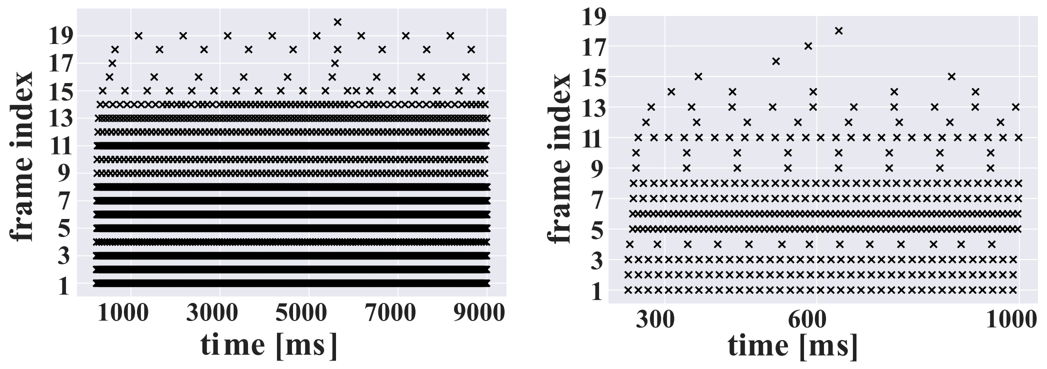


Figure 9.3: An example of the times of transmission of K_b is shown. There each dimension represents the type of transmitted frames and each data point its point of occurrence. The right part shows a magnified version of the left plot. It can be seen that both cyclic, as well as event based frames are transmitted.

	ACC	Light	Wiper	Startup	Shutdown
# TVs	26	78	32	41	441
# samples [min/max/avg]	44268	1725447 [2/110418/8584]	387462 [2/105828/3761]	1372079 [2/112070/5991]	49950 [2/7064/2378]
# journeys	1	9	12	9	1
Duration s [min/max/avg]	5106,508	0,49/28540,26 /3070,18	0,02 / 19322,75 / 2698,95	0,07 / 28540,26 / 2351,68	10,74/4007,32 /1431,77
mean gap s	0,115	3,554	2,643	11,721	1,71
std gap s	0,622	7,796	9,221	14,517	7,14

Table 9.1: The statistics of each data set are shown. The type of data corresponds to $\mathbf{K}_{cond}$ of the presented pipeline.

behavior during the wiping procedure.

As presented in Chapter 4 initial traces $\mathbf{K}_b$ are frames that are transmitted between ECUs over time. There, each frame contains a set of TV information. An example of such a trace at this stage is shown in Figure 9.3.

9.4 Mining in-vehicle Network Traces for Specifications

In this section the results of the case study are presented. Please refer to the full evaluation results given in the appendix, where results of all data sets of all steps are given. In this chapter only a relevant subset of results is shown and discussed to illustrate all relevant aspects of the DM pipeline.

9.4.1 Preprocessing

After preprocessing the frames $\mathbf{K}_b$ result in MESs $\mathbf{K}_{cond}$ and MSSs $\mathbf{K}_n$. The statistics of each data set for $\mathbf{K}_{cond}$ is shown in Table 9.1 and Figure 9.4. Statistics of $\mathbf{K}_n$ are provided in Table 9.2 and Figure 9.5. The data sets were chosen to be of different shape

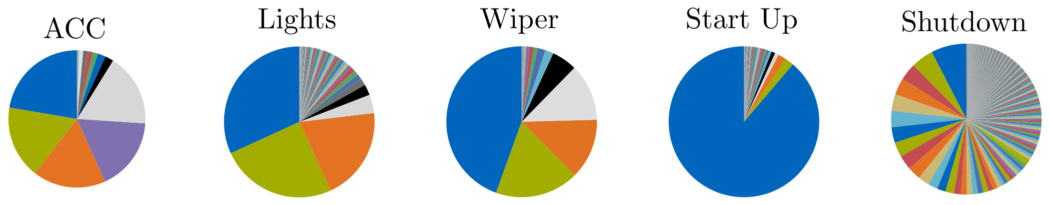


Figure 9.4: The distribution of TVs for K_{cond} is shown. Each color represents one TV and its number of occurrences in the data set.

in terms of TV distribution, number of samples and gap distributions as can be seen in Table 9.1, where ACC, Wiper and Start up have less TVs, Light a medium number and shutdown a high number of TVs. Also, the number of samples and the duration vary across those data sets. While ACC and shutdown are smaller with around 50 000 samples, a medium number of samples of around 400 000 samples is found in Wiper and a high number of samples with around 1.5 million samples is given in the other data sets. Further, in Figure 9.4 it can be seen that there are some TVs that are dominating the data set before preprocessing, which are often numerical samples as those are changing in value more often. The preprocessing introduced in Chapter 4 is applied.

Effect of Preprocessing: The result of preprocessing is a more homogeneous data set which is preferable for Specification Mining. This is due to the fact that otherwise more frequent TVs are preferred as those are present in more segments even if less correlation is given. This would blur precision of learned specifications. Homogeneity is improved as the number of numerical TVs is reduced. As Figure 9.5 illustrates this yields more uniformly distributed TVs. For sets with more numerical TVs, such as the Lights set, this is more evident, while it changes less for less numerical TVs, such as in the ACC case.
Similarly, for the same reason the reduction in sample number is stronger in the further case, with reduction rates of 80 % for Light, 74 % for Wiper, 55 % for Start up but only 2 % for ACC and 3 % for Shutdown. This reduction is highly desirable and consequently allows to apply the further steps of the DM pipeline, with the biggest data set size being around 600 000 samples after preprocessing.
Above that the preprocessing on average increases gap sizes between consequent samples, e.g. in lights the gap size grows from 3.5 to 7.8 seconds. That is, samples that are correlated are still within close proximity, but numerical samples that often changed in value before reduction (resulting in false positive correlations) are only in proximity if those change significantly. This is especially important, as with this the range segmentation approach is able to separate functional procedures more easily as bigger gaps are present between such procedures and the range per sample determines separation.

Resulting distribution per TV: The preprocessed data sets, that are used as input for the further steps, vary in shape as Figure 9.5 shows. ACC has mostly nominal TVs where 5 nominal TVs are dominant. Numerical and ordinal TVs occur equally often. Lights has mostly nominal TVs (ca. 70 %), with less numerical TVs (ca. 20 %) and TVs of remaining types (ca. 10 %). There, for all types a subset of TVs is more dominant.

	ACC	**Light**	**Wiper**	**Startup**	**Shutdown**
# TVs [num /nom/bin/ord]	26 [4 /20/0/2]	78 [14 /38/21/5]	32 [8 /19/2/3]	41 [3 /35/3/0]	441 [14 /301/79/47]
# samples [num /nom/bin/ord]	43098 [1553 / 41533/ 0/2]	344948 [75581 / 231132/ 24497 /13738]	98109 [75347 / 12799/ 6481/ 3482]	616661 [477796 / 134958/3907/0]	48604 [2202 / 257 44/8801/11856]
mean gap s	0,118	7,826	25,11	12,488	1,725
std gap s	0,646	16,374	34,644	17,087	7,252

Table 9.2: The statistics after preprocessing are shown, which corresponds to $\mathbf{K}_n$.

For the Wiper case, numerical TVs (ca. 77 %) are dominant, with less values of nominal (ca. 13 %), ordinal (ca. 3 %) and binary type (ca. 7 %). Here, for each case one TV dominates per type. A similar distribution is given for the Start up set with 77 % numeric, 21 % nominal and 2 % binary TVs. The Shutdown set has a more evenly distributed set of TV types, with 52 % being nominal, 25 % ordinal, 18 % binary and 5 % numerical. Here, nominal TVs have many TVs that occur equally often while some are transmitted only little, while in the other cases again some TVs are dominantly present.

Conclusion: For all data sets a homogenization and reduction of data was achieved by applying the automated preprocessing approach. This was demonstrated here for big data set sizes of up to 1.8 million samples. Also, the given data set contains dominant TVs, which are often either numerical or nominal and resemble dominant dimensions of functional procedures. Less occurring samples often transmit less frequent state changes of the car (such as driving or parking) that are not directly part of functional procedures. However, in most cases such TVs are still relevant to be considered in those procedures. But, in some cases those TVs only transmit information ones (e.g. type of the car) which is mostly done by TVs that are not relevant for functional procedures. In the following those data sets are used to show the working of the proposed DM pipeline for Specification Mining and dominant behavior identification.

9.4.2 Clustering

Next, clustering of TVs is performed on each data. This includes a hyper parameter estimation which allows to choose the best parameters per data set. For the optimal parameters the resulting clustering is characterized. Notably, clusters of size one are considered noise here and are assigned to one noise cluster. Further, the MES trace K_{cond} is ingested here.

Results of hyper parameter estimation: For each data set a grid of $\epsilon = [0.1; 7.0]$, $MinSamples = [2; 9]$ and number of PCA components $n_{PCA} \in \{5, 10, 15, 20, \emptyset\}$, where $\emptyset$ corresponds to no PCA, is evaluated based on the Silhouette index. The resulting values of the metrics are shown in Figure 9.6 for the ACC, in Figure 9.7 for the Lights data set and in the Appendix for the further data sets.
The heat map in those figures shows the results for different ϵ, $MinSamples$ and n_{PCA}. It can be seen that with $n_{PCA} = 5$ for the lights data set larger ϵ and smaller $MinSamples$ improves the index. That is as too small ϵ tend to form clusters of size one, while bigger clusters grow with bigger values. Also, a higher number of $MinSamples$

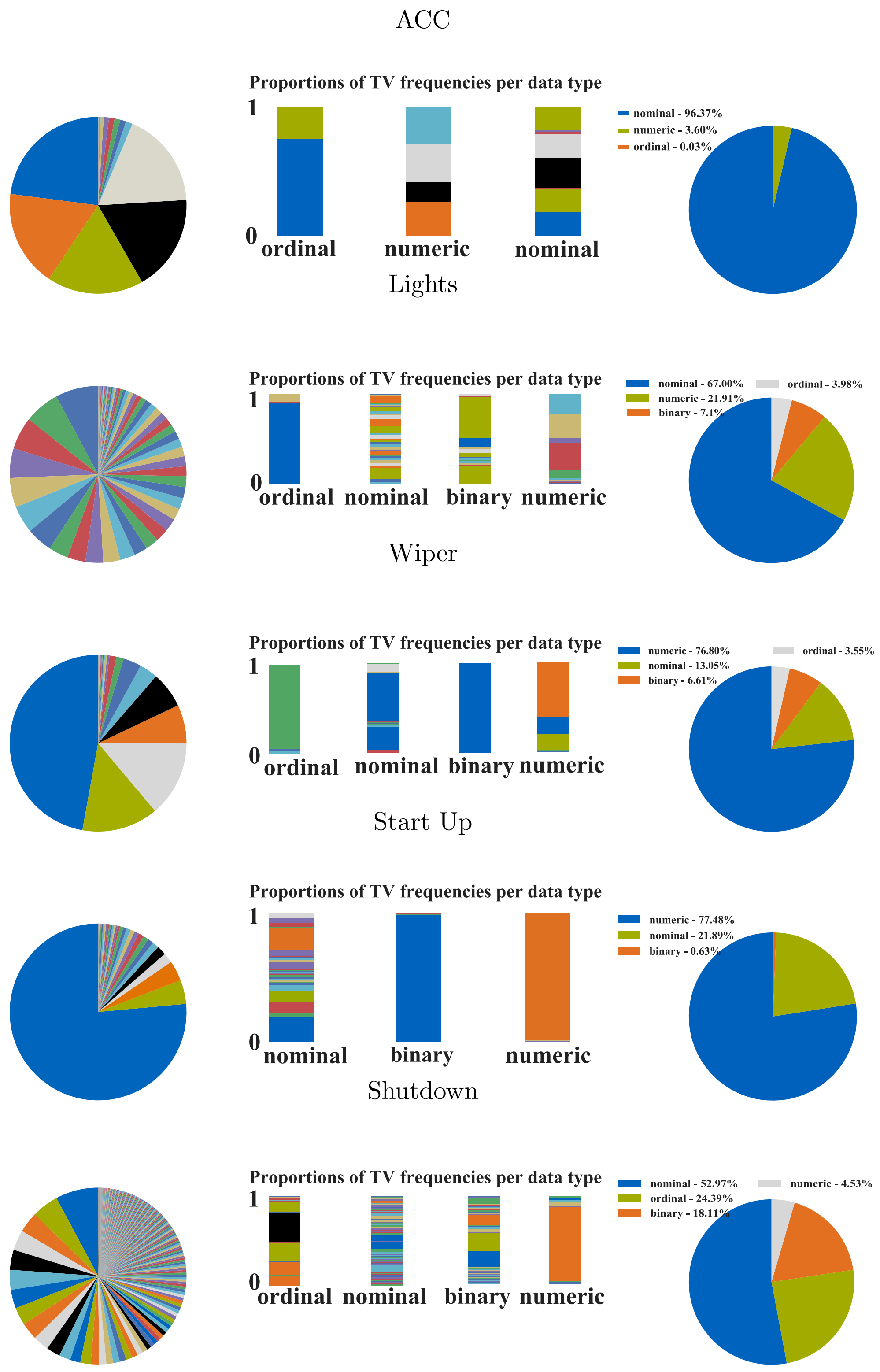


Figure 9.5: Here the statistics of all data sets are shown after preprocessing, i.e. $\mathbf{K}_n$. There the left pie chart shows the numbers of occurrences per TV, the middle chart show the same distribution per data type and the right plot illustrates the distribution of data types.

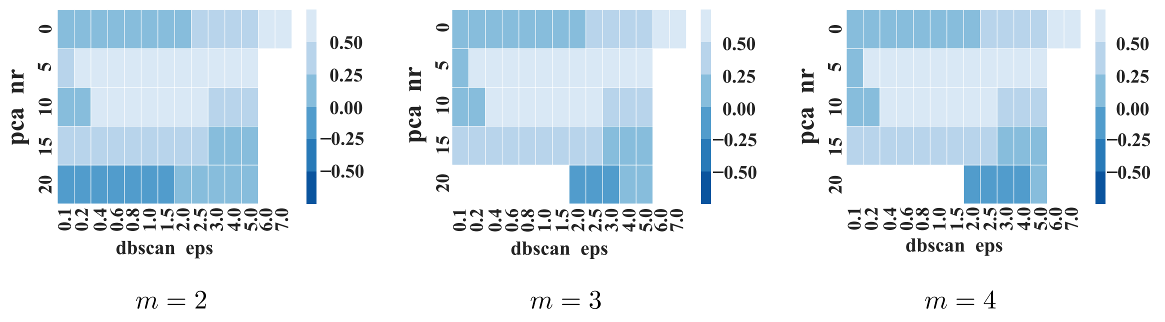


Figure 9.6: Results of hyper parameter estimation for TV clustering of ACC. The color indicates the value of the Silhouette index for various number of PCA components and for various ϵ values, for different values of $MinSamples$ m.

creates more TVs that are considered noise, which excludes possibly consistent clusters of smaller size resulting in worse values for the Silhouette index. Further, the heat maps show white spots. Those occur if all data points are considered noise, e.g. if no cluster bigger than $MinSamples$ is found.
The effect of increasing ϵ and $MinSamples$ is shown in Figures 9.8 and 9.9 for the lights data set if the respective other values are set to its optimal choice.
An optimal parameter has a maximal Silhouette index, excludes as few data points as possible and results in more clusters of sufficiently large size. Here, clusters smaller than three are assumed meaningless, i.e. $MinSamples > 2$. Hence, $n_{PCA} = 5$, $\epsilon = 1.5$ and $MinSamples = 3$ give the best results for the Lights data set in that case.

Choice of parameters: Similarly, for the other data sets best parameters are the following. For ACC that is $MinSamples = 3$. $n_{PCA} = 5$ and $\epsilon = 1.5$ giving an index of around 0.5, for Lights that is $MinSamples = 3$. $n_{PCA} = 5$ and $\epsilon = 1.5$ giving an index of around 0.5, for Wiper that is $MinSamples = 3$, $n_{PCA} = 5$ and $\epsilon = 1.5$ giving an index of around 0.6, for Start up that is $MinSamples = 3$. $n_{PCA} = 5$ and $\epsilon = 1.0$ giving an index of around 0.4 and for Shutdown that is $MinSamples = 3$. $n_{PCA} = 5$ and $\epsilon = 0.8$ giving an index of around 0.7.
Thus, according to the index, best separation is given in the Shutdown case. This might be as this set has the highest number of TVs, making the results statistically more significant. Also, the shutdown case contains many TVs of different functional procedures as it was not preselected at all, while in the other four cases a preselection was performed leaving a set of various functional procedures that are less strongly correlated than in the Shutdown case. Another reason is that for a smaller window size (here 400) a better separation is found if less samples are given as common occurrences are found on a finer granularity. In particular, as durations are in a similar range (1500 to 3000 seconds per journey), this granularity is characterized by the mean and standard deviations of gap size between consequent elements. That is, a higher standard deviation to mean ratio is expected to yield a bigger spread of data points and thus, better separability. However, if this spread is too high the data points are expected to be less frequently found in common windows and thus, to give worse separation. This ratio is 4.2 for shutdown

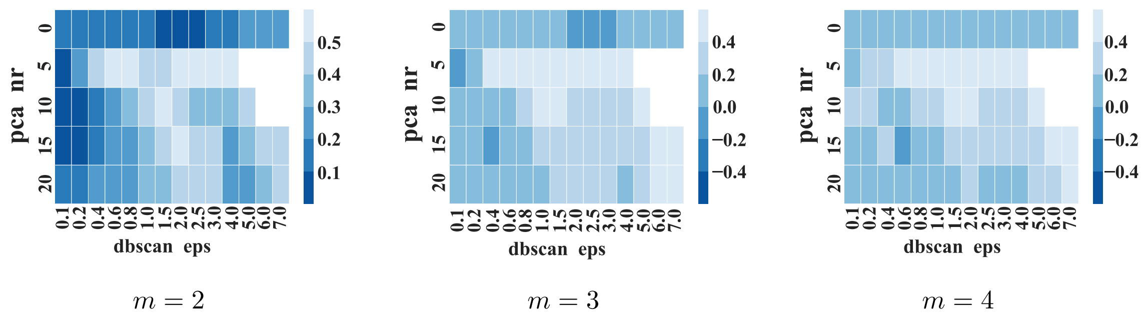


Figure 9.7: Results of hyper parameter estimation for TV clustering of Lights. The color indicates the value of the Silhouette index for various number of PCA components and for various ϵ values, for different values of $MinSamples$ m.

(index ca. 0.7), 24.3 for Wiper (index ca. 0.6), 5.4 for ACC (index ca. 0.5), 2.2 for lights (index ca. 0.5), 1.2 for Start up (index ca. 0.4). The Start up set has the lowest ratio which confirms the expected outcome. When excluding the Shutdown set this is also valid for the other data sets. Shutdown does not follow this pattern. This is as in comparison to the other data sets it has significantly more TVs, while having less samples.
However, still for all cases 400 windows per journey and an ϵ value between 0.8 and 1.5 results in clusterings with good Silhouette index between 0.4 and 0.7, as well as a clustering that is meaningful, which is discussed next.

Found TV clusters: For ACC three clusters were found which were named by analyzing the grouped instances. These are named as (1) display information (e.g. TVs *display active*, etc.), (2) car state information (e.g. TVs *brake state*, *speed*, etc.) and a fragmentary cluster. Clusters of the lights data set could be named as (1) light behavior including states of the indicator or front lights (e.g. TVs *handle bar state*, *left indicator state*, etc.), (2) detection of traffic (e.g. TVs *traffic detected*, etc.) (3) engine movement (4) environment values and a fragmentary cluster. Clusters of the Wiper data set could be named as (1) rain sensor and wiper control (e.g. TVs *wiper activation*, *wiper speed*, etc.), (2) wiper position and a fragmentary cluster which includes (3) environmental information. Clusters of the Start up data set could be named as (1) engine and energy activation (e.g. TVs *engine running*, etc.), (2) car state and energy (e.g. TVs *subsystem control*, *engine state*, etc.) and two clusters (3) and (4) with 3 TVs each with no meaningful description. In the Shutdown data set most dominant clusters could be named as (1) car state information before shut down (e.g. TVs *engine state*, *engine component*, etc.), (2) state information after shutdown (e.g. TVs *start button state*, *engine state*, etc.) or (3) key information.

Conclusion: As already discussed in chapter 5 TVs can be clustered based on common occurrences yielding smaller subgroups of TVs that typically relate to common functional procedures. However, this excludes TVs that change rarely, but might be relevant to a functional procedures. That is why in the following, instead of using the discovered

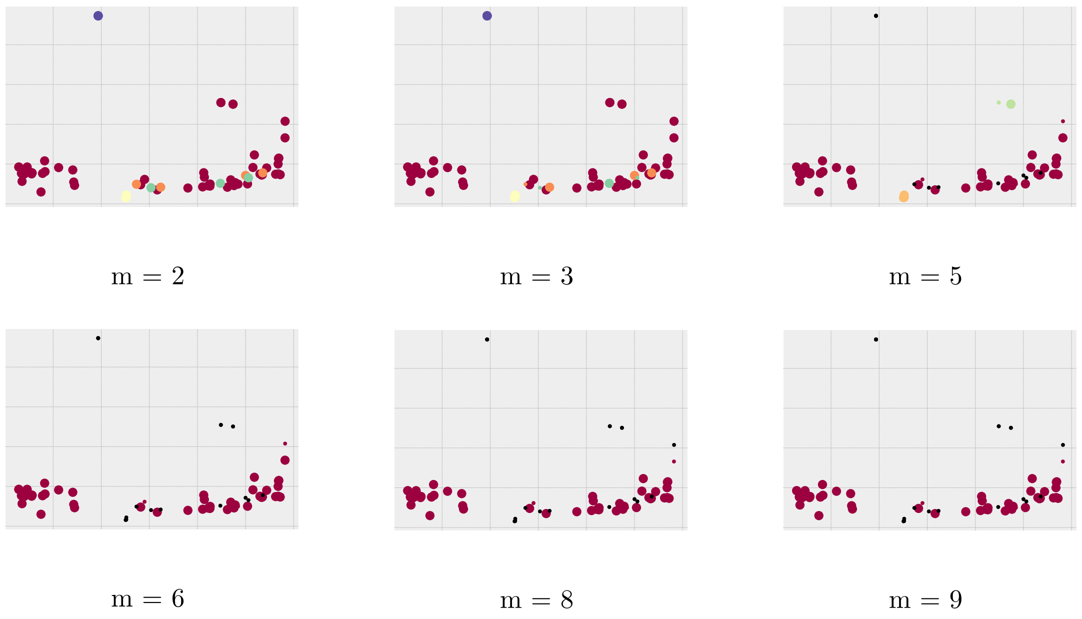


Figure 9.8: Results of increasing *MinSamples m* when clustering TVs in the Lights data set, when considering the first two PCA components, where $n_{PCA} = 5$ and $\epsilon = 1.5$.

clusters directly an expert used the resulting clustering to determine a subset of TVs which is relevant per data set. The chosen data sets which are used in the following are characterized in the appendix. For ACC one such cluster is determined, for Lights two clusters are determined, for wiper one cluster is determined, for Start up two clusters are determined and for Shutdown one cluster is determined. Those are named as *TV Cluster i* where i is the TV cluster index.
Further, all data sets have a similar average duration of around 1500 to 3000 seconds per journey. We chose 400 overlapping windows for clustering, which was sufficient to discover correlations. Best separation is achieved, if a significant number of TVs is present, the spread of data points is in a good range and TVs differ stronger in terms of times of occurrence, as exemplified with the Shutdown case.

9.4.3 Segmentation Clustering

Based on the reduced clusters that were created by experts, the range segmentation is used to identify segments.
For the ACC data set all TVs are used and clusters include the activation and deactivation of the ACC function, which is found by segmenting the trace at a fixed time range before those states (referred to as *targeted method* here). In the following it is distinguished between the ACC data set which uses this targeted method and all other data sets which use range segmentation. As described in chapter 6 in this method segments are found by specifying a temporal range around each state change and by grouping close state changes. Hence, for the ACC case hyper parameters are the temporal threshold before the state change t_{prev}, while for range segmentation parameters are the range

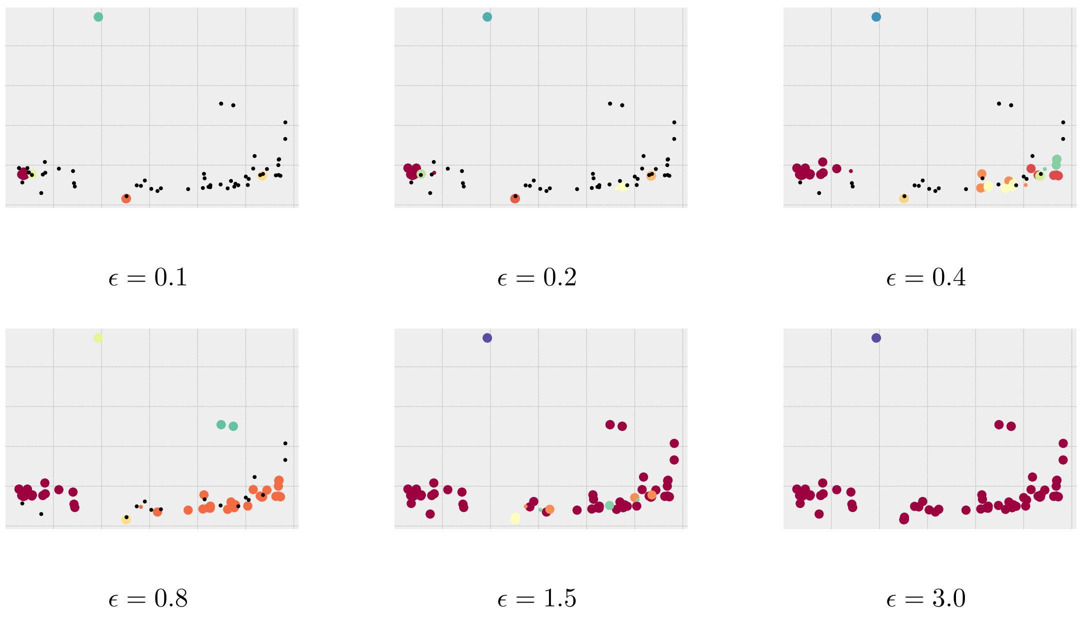


Figure 9.9: Results of increasing ϵ when clustering TVs in the Lights data set, when considering the first two PCA components, where $n_{PCA} = 5$ and $MinSamples = 3$.

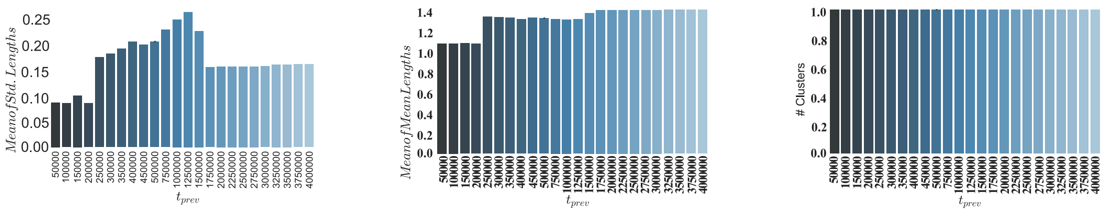
around each data point r_{temp}, as well as ϵ for clustering.

Metrics: Here, four metrics are considered for evaluation. First, the mean sequence length. Clusters of short length tend to be more consistent, while those are meaningless to consider as functional procedures. Therefore, larger values of this metric are preferred. Second, the standard deviation of the mean sequence lengths is considered. That is, if all sequence lengths across all extracted segments are of identical length per TV, it is very likely that a consistent segment set is found, which in turn results in a low value for this metric. In contrast to that, if sequences vary in length per TV this value is high. Third, the number of found clusters is considered here. This value indicates the granularity of clustering, i.e. if multiple fragmentary clusters are found or one big cluster is found. Both extremes are not desired. Rather a reasonable number of clusters that depends on the number of samples is to be chosen. Lastly, the Silhouette Index is considered as it indicates the separation of clusters in terms of variation in TV occurrences. Here, a higher value is preferable. To evaluate the influence of parameters in those segmentation approaches for the given scenarios, those given hyper parameters were varied for each data set.
To cover both approaches hyper parameter performance is discussed for the ACC set and both for the Lights and Start up data set, as similar conclusions can be drawn for the remaining data sets.

Results of hyper parameter estimation: First, the results for the ACC data set are discussed, which are presented in Figure 9.10. Here the activation is considered only,

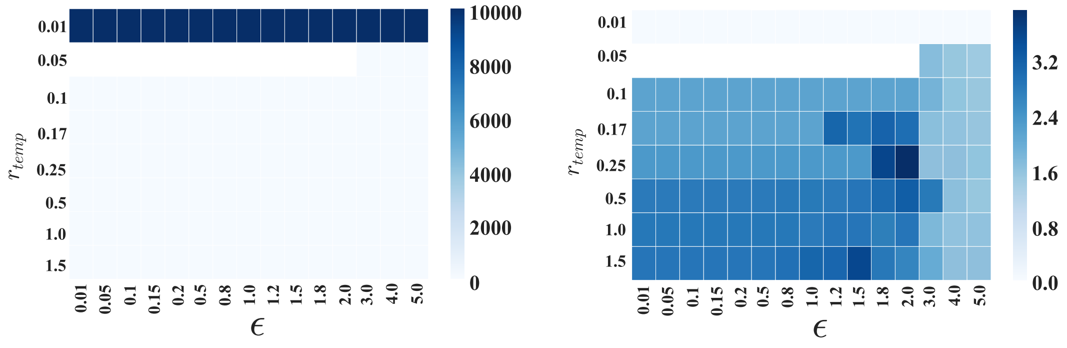
as the deactivation results in similar conclusions. Looking at the left chart in Figure 9.10 it can be seen that for lower ranges the standard deviation is low. In this area only less state changes are given, which is non-informative when considering functional procedures. With increasing values the deviation grows up to 1.5 seconds before it drops significantly. The increase results from noisy state changes that are added, while the drop comes when a further set of state changes is found that is common in all segments, which seems to be the case here. Thus, at this drop point meaningful functional procedures are likely to be found. Also, the mean length of sequences per TVs grows and is at its highest point at the drop point. The number of clusters is one, as all activations form one cluster.
Second, the results of the Lights data set when choosing *TV Cluster 1* are discussed. Those are illustrated in Figures 9.11 and 9.12. It can be seen that for small $r_{temp} = 0.01$ and $r_{temp} = 0.05$ no meaningful sequences are found (indicated by dark blue in the standard deviation plot, white in the means plot and white in the number of clusters plot). That is, no found segment group contains the minimum required number of sequence elements to form a valid cluster. That is, as at each point's neighborhood no overlapping data point ranges are given. However, with increasing range size more and more segment groups fall within a valid area of length, giving a growing number of clusters. At the same time, the length of sequences per TV grows as it becomes more likely for state changes of a certain TV to be contained within the same segment group.
ϵ defines the distance (in terms of similar numbers of TVs per group) at which multiple segments are clustered together. With increasing value more segments fall in the same cluster. With this, the composition of segments within the clusters changes, yielding a change in value of mean sequence lengths. Also, the number of clusters becomes smaller as close groups are fused with growing ϵ. As the Silhouette index shows, there is no improved separation for $r_{temp} > 0.1$, while this index becomes worse when choosing a too high value for ϵ. For similar number of clusters and Silhouette indices it is preferable to chose sequences that are longer on average, which is why for *TV Cluster 1* in the lights data set a value of $\epsilon = 0.5$ and $r_{temp} = 0.5$ are chosen.
Third, the results of the Start up data set when choosing *TV Cluster 2* are discussed. This is as here an effect that is different to the Lights data set is observed. As Figures 9.13 and 9.14 show, in the first part the number of clusters grows with increasing r_{temp} and decreases then. The increase is again due to more segments with minimum length, while the decrease results from more data points that are overlapping forming longer sequences that are fused to one cluster. This can be seen as in those cases the standard deviation and mean sequence lengths grows, and the Silhouette index decreases. Increase of ϵ behaves similar to the Lights case. We chose an $\epsilon = 0.01$ and $r_{temp} = 0.1$ here to keep the mean lengths of TVs in a reasonable range (note that lengths of up to 40 state changes per TV were found).

Choice of parameters: Based on above discussion the following parameters are chosen for the given data sets. Those are for ACC TV Cluster 1 segment group 1 $t_{prev} = 2.0$, for ACC TV Cluster 1 segment group 2 $t_{prev} = 2.0$, for Lights TV Cluster 1 $\epsilon = 0.5$ and $r_{temp} = 0.5$, for Lights TV Cluster 2 $\epsilon = 1.0$ and $r_{temp} = 0.01$, for Wiper TV Cluster 1 $\epsilon = 0.8$ and $r_{temp} = 0.1$, for Start up TV Cluster 1 $\epsilon = 0.01$ and $r_{temp} = 0.1$, for Start up TV Cluster 2 $\epsilon = 0.01$ and $r_{temp} = 0.5$, for Shutdown TV Cluster 1 $\epsilon = 1.0$ and



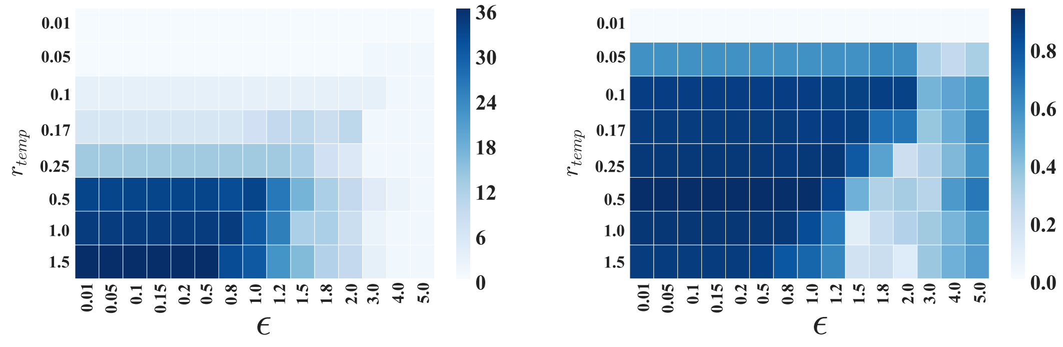
Mean Std. of Seq. Lengths Mean Seq. Lengths Number of Clusters

Figure 9.10: Hyper parameters for ACC, when varying t_{prev} in terms of various metrics.



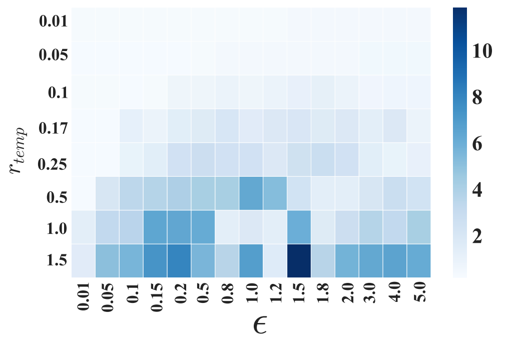
Mean of Std. of Sequence lengths per TV Mean of Sequence lengths per TV

Figure 9.11: Hyper parameters for Lights, when varying r_{temp} and ϵ in terms of various metrics.

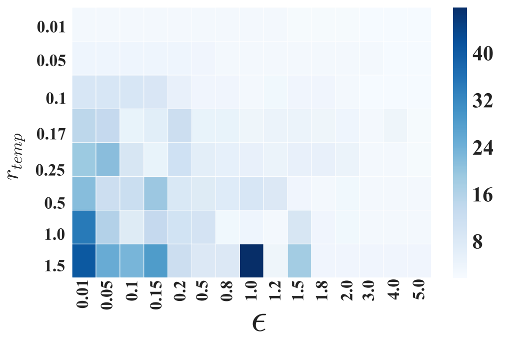


Number of Clusters found Silhouette index

Figure 9.12: Hyper parameters for Lights, when varying r_{temp} and ϵ in terms of various metrics.

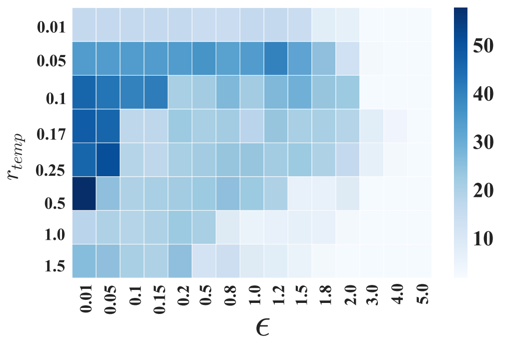


Mean of Std. of Sequence lengths per TV

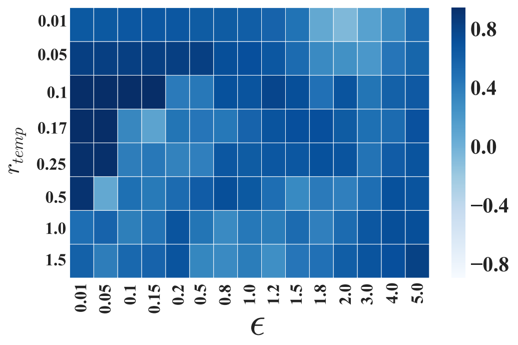


Mean of Sequence lengths per TV

Figure 9.13: Hyper parameters for Start up, when varying r_{temp} and ϵ in terms of various metrics.



Number of Clusters found



Silhouette index

Figure 9.14: Hyper parameters for Start up, when varying r_{temp} and ϵ in terms of various metrics.

$r_{temp} = 0.25$ and for Shutdown TV Cluster 2 $\epsilon = 0.01$ and $r_{temp} = 0.1$. Further, the clustering was performed with 10 PCA components.

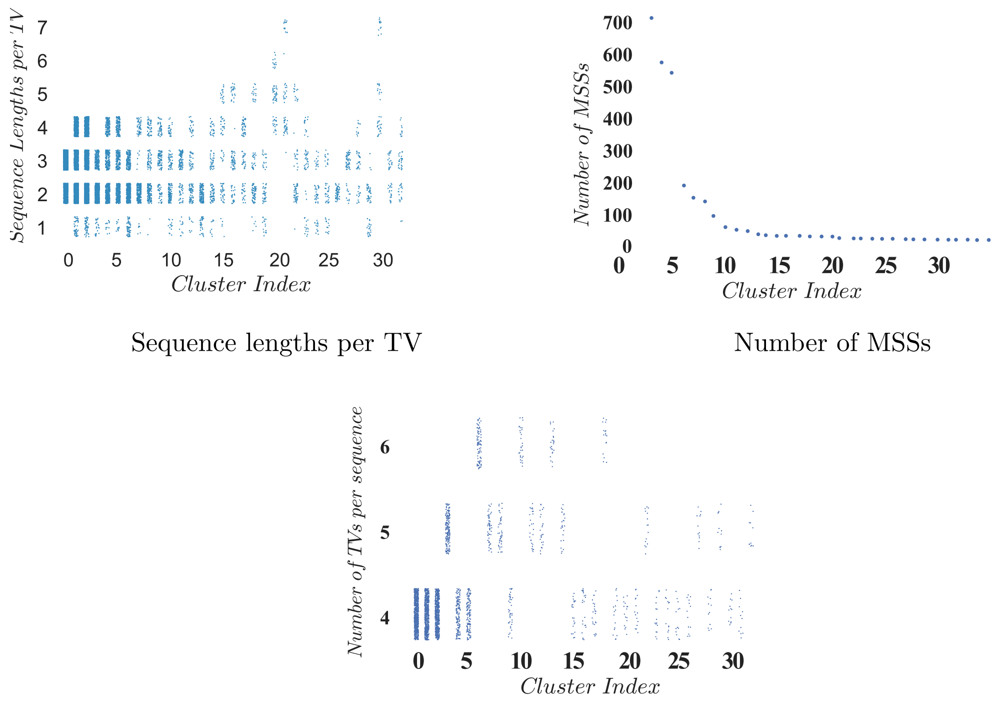
Found segment clusters: With those parameters meaningful MSS sets are found. In particular the range segmentation excludes both MSS groups of too short length as well as of to small number. Depending on the parameterization multiple MSS groups might resemble similar behavior, that was not grouped together due to different types of noise that was present. In the future such clusters might be grouped together by applying post processing. Further, the choice of parameters resulted in a rather fine grained clustering as ϵ was mostly chosen as low as possible. With this, less variation in segments is present, which on the one hand is desirable in Specification Mining but on the other hand results in correlations that are hard to compute during structure discovery. This will be discussed further below.
Some of the found MSSs might be described as follows, where the following argumentation is based on the statistics of those sets. For TV Cluster 1 of the Lights data set those statistics are shown in Figure 9.15, while for the remaining sets those can be found in the Appendix. The same hold for examples of found clusters which are depicted in Figure 9.16 and 9.17 for the lights data set.

- **ACC:** Here, two MSS groups with mainly 11 involved TVs are found which is the functional procedure before activation and the behavior before deactivation of the ACC.

- **Lights:** For TV Cluster 1, 32 MSS groups with up to 7 TVs were found, where 3 groups are dominant with more than 500 MSSs and further 3 groups with more than 100 MSSs. Those groups include the activation of the indicator for 3 seconds, the permanent activation of the indicator and the starting of the indicator in one direction and changing it directly to the other. Other clusters are variants of this. Further, it can be seen that the number of MSSs tends to decrease with the length of MSSs. This is as MSS groups with longer sequences TVs have a higher risk of noise occurring in between. This noise consequently results in MSSs that are grouped in different clusters. For TV Cluster 2 28 MSS groups with up to 10 TVs were found, where 2 are dominant with more than 500 MSSs. Those groups include the movement of lights on occurring traffic, the automated activation of the high beam, the light state depending on the driving mode, the variation of brightness, the detection of traffic or combinations of those.

- **Wiper:** Here 68 MSS groups with up to 9 TVs are found. The dominant group contains more than 1000 MSSs and resembles the cyclic movement of the wiper. Notably, with this the range segmentation was able to detect MSSs with cyclic movement. Further groups include the manual and automated activation and deactivation of the wiper. Further, as this data set has many discretized continuous values multiple clusters with variations of those values are found, such as the rain intensity or brightness. Thus, in comparison to all other data sets lengths of sequences per TV are up to 60 with many in a range around 10.

- **Start up:** For TV Cluster 2, 41 MSS groups with up to 8 TVs were found, where 3 groups are dominant with more than 500 MSSs and further 5 groups of above



Sequence lengths per TV

Number of MSSs

Figure 9.15: Statistics of TV Cluster 1 of the lights data set are shown here. Including the distribution of sequence lengths per TV, the number of MSSs per cluster and the number of TVs per sequence.

100 MSSs. Here, 20 of all found groups include only one TV. Those are again variations of continuous TVs that are grouped, which also can be seen as those clusters are longer. Such groups are meaningless and are excluded in the further process. Other clusters range from 2 to 8 TVs. Those found clusters include activations and deactivations of subsystems and the power supply, as well as the successive activation and deactivation of the engine. For TV Cluster 3, 17 MSS groups with up to 7 TVs were found, which all occur less than 80 times. Here no continuous TV is present and thus, all MSS groups resemble meaningful behavior. Found clusters include similar activations and deactivations as in TV Cluster 2, but with a different set of TVs considered.

- **Shutdown:** For both TV Cluster 1 and for TV Cluster 2 2 MSS groups were found. That is, as here an explicit filtering of the trace that contains such shutdown procedures was performed beforehand. Thus, the resulting behaviors are similar. Up to 5 TVs are present here. Found segments are similar to the Start up case and include similar activations and deactivations, but with a different set of TVs considered.

Conclusion: The above discussion shows that both range and targeted segmentation are capable to automatically find meaningful MSS groups when using suitable hyper parameters. Based on this, subsequent analyses are performed per functional

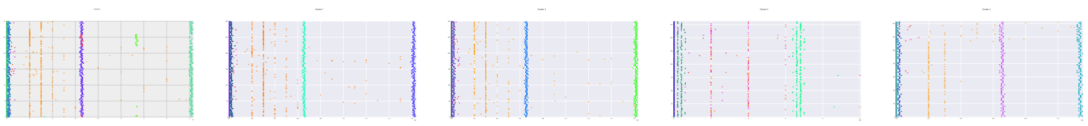


Figure 9.16: MSS groups 0 to 4 are shown for the lights data set.

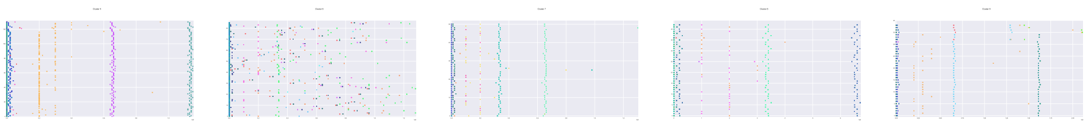


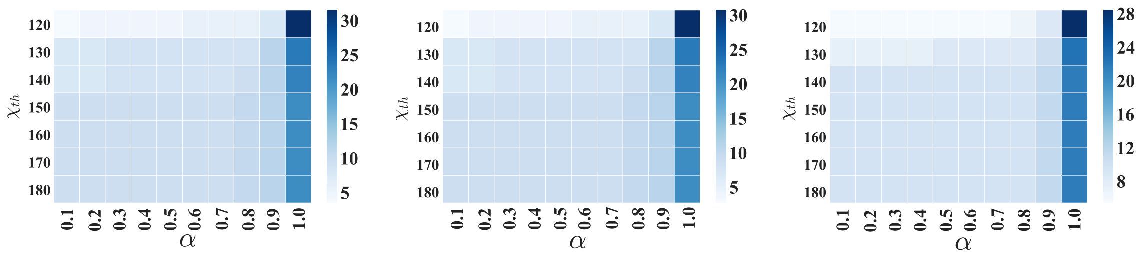
Figure 9.17: MSS groups 5 to 9 are shown for the lights data set.

procedure that allow to extract specifications for defined scenarios. Further, for the inspected data sets suitable parameters seem to be within a range of $r_{temp} = [0.01, 0.5]$ and $\epsilon = [0.01, 1.0]$. Data sets with more continuous TVs tend to find longer MSS groups, which however, often are restricted to less TVs. Such groups might require different specification extraction approaches that focus on cyclic behavior. Above that, groups with longer MSSs are smaller due to variation in noise, as parameters where chosen to find groups of high segment consistency. In the following, a subset of found clusters is used to demonstrate how specifications are extracted from those MSS groups.

9.4.4 Structure Discovery

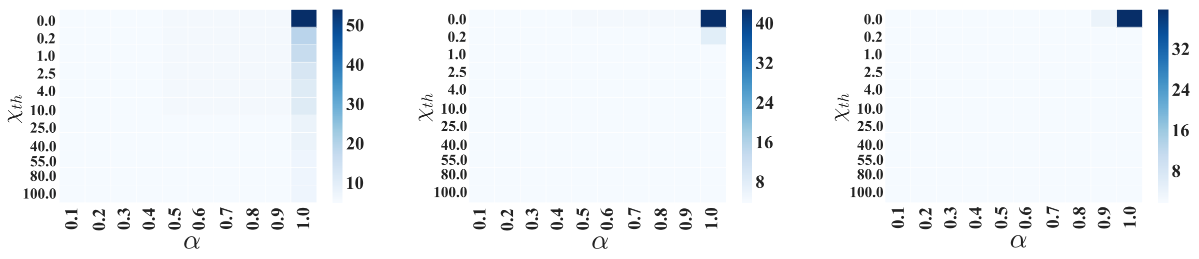
Multiple MSS groups, that resemble distinct behaviors, were evaluated in terms of the influence of parameters on the discovered structures. For this the number of inter edges was considered as metric as it best resembles the complexity of the learned structure. A low number indicates no edges and thus, a low model capacity. Less potential paths through the model exist, that are passed with BaySpec. At worst this leads to specifications that include a single TV. In the opposite case of a high number of edges a high number of redundant paths is given in the network. However, this is preferable here, because still parameter estimation will parameterize the network such that the actual observed behavior is more likely in the network. The same holds for the extraction of dominant behavior using MPE, as during sampling observed constellations will be more likely. Here, structure learning of the activation in ACC, the indicator activation of the Lights and the first MSS group of the Start up set is discussed. Again the remaining results are found in the appendix.

Results of hyper parameter estimation: For the following discussion it is important to note, that two types of MSS groups were created with the previous approaches. Using range segmentation the goal was to find consistent MSS groups. In contrast to that using the targeted approach no consistency is guaranteed and noise might be included. Further, k defines the level of filtering of inconsistent MSSs and both χ_{th} and α measure correlation between two state changes.
First, when looking at the ACC case in Figure 9.18 it can be seen that with a higher k the number of inter-edges decreases. This is as a growth of this parameter, results in



$k = 0.00$ $k = 0.05$ $k = 0.15$

Figure 9.18: Number of inter edges for different parameters k, χ_{th} and α for the ACC data set (activation)



$k = 0.00$ $k = 0.15$ $k = 0.35$

Figure 9.19: Number of inter edges for different parameters k, χ_{th} and α for the Lights data set (TV Cluster 1, MSS group 0)

more filtering and thus, less distinct MSSs. Further, it can be seen that the highest number of inter-edges is given if less correlation is required (i.e. χ_{th} is small and $\alpha = 1.0$). With increasing χ_{th} and decreasing α less inter-edges are given as edges with lighter correlation are dropped. However, for the ACC case still up to around 10 inter-edges are found if α is chosen low.
For the Lights and Start up set in Figures 9.19 and 9.20 this effect becomes more evident. For both cases an $\alpha < 0.9$ significantly reduces the number of inter edges as those data sets are high in consistency and thus, barely usable for computation of correlations. Nevertheless, TrieDiscover bases the learned structure on the temporal occurrences of state changes, which leads to consistent structures even if no correlation computation is possible. This is desirable as with this BaySpec and MPE are able to perform inference from the structure even if redundant connections are present.

Choice of parameters: As stated before, a higher number of inter-edges is preferable. Thus, parameters were chosen accordingly. For all data sets k was set to 0. For the three above cases the following parameters were chosen: the activation case of ACC $\chi_{th} = 0.2$ and $\alpha = 1.0$, for Lights TV Cluster 1 group 0 $\chi_{th} = 0.2$ and $\alpha = 1.0$, for Start up TV Cluster 2 group 1 $\chi_{th} = 0.0$ and $\alpha = 1.0$.

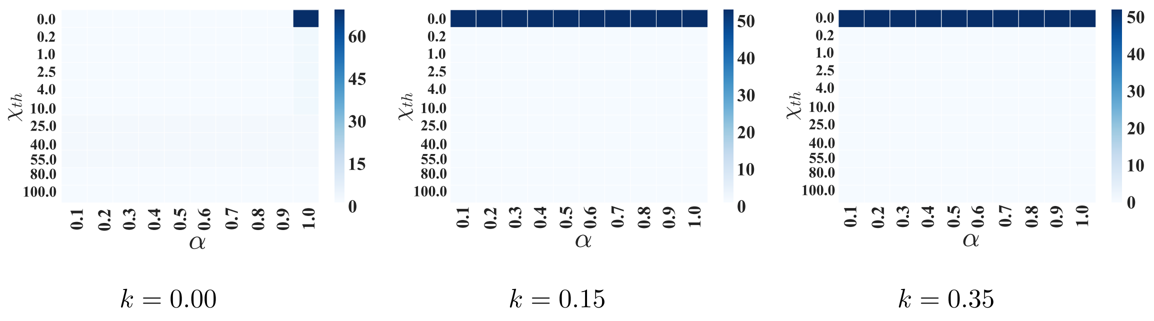


Figure 9.20: Number of inter edges for different parameters k, χ_{th} and α for the Start data set (TV Cluster 2, MSS group 1)

Found structures: The structures found for the three cases above are shown in Figures 9.21, 9.22 and 9.23. As shown in Figures 9.21 ACC has less nodes, i.e. state changes, occurring prior to activation. Rather often a single state change or a constellation of state changes that is represented in the initial nodes forces the deactivation. Consequently, MPE is expected to be more insightful when inferring the activation behavior. The structure learned for the lights indicator also seems plausible, as multiple state changes in change of handle bar and activation of indicators occur per activation as Figure 9.22 shows. The segment plot also shows that there are variants of this behavior including defect activations, as well as right and left activation.
An example for a structure of higher complexity is given in the Start up case in Figure 9.23. In this case occurring MSSs mostly vary in their initial state: However, after any initial state combination, a similar set of actions occurs, as the segments plot shows.

Conclusion: While correlation is hard to compute if samples of MSSs are too consistent, TrieDiscover is still able to extract meaningful structures. However, those structures tend to have a too optimistic number of inter-edges. This is not a problem for the scenario of specification extraction within the proposed pipeline since the subsequently executed methods BaySpec and MPE are able to extract the dominant behavior, which is prevalent when learned from observed samples.
In the future, this number of inter-edges could be reduced by making correlations more evident or by developing different correlation metrics. This could be achieved by merging MSS groups that resemble similar (but not identical) behavior with more variations.

Parameter Estimation: As was shown in Chapter 7 all parameter estimation approaches for TSCBNs perform similar. Thus, for all given cases EM or MPE-R were used. With this, the parameters were learned and a fully parameterized structure per MSS group is extracted.

9.4.5 Specification Mining

Given the parameterized models, BaySpec and MPE were applied to extract specifications and dominant behavior. As metrics the width and height is used as defined in Chapter 8. Further, the metric-based approach is used. There the literal ratio and com-

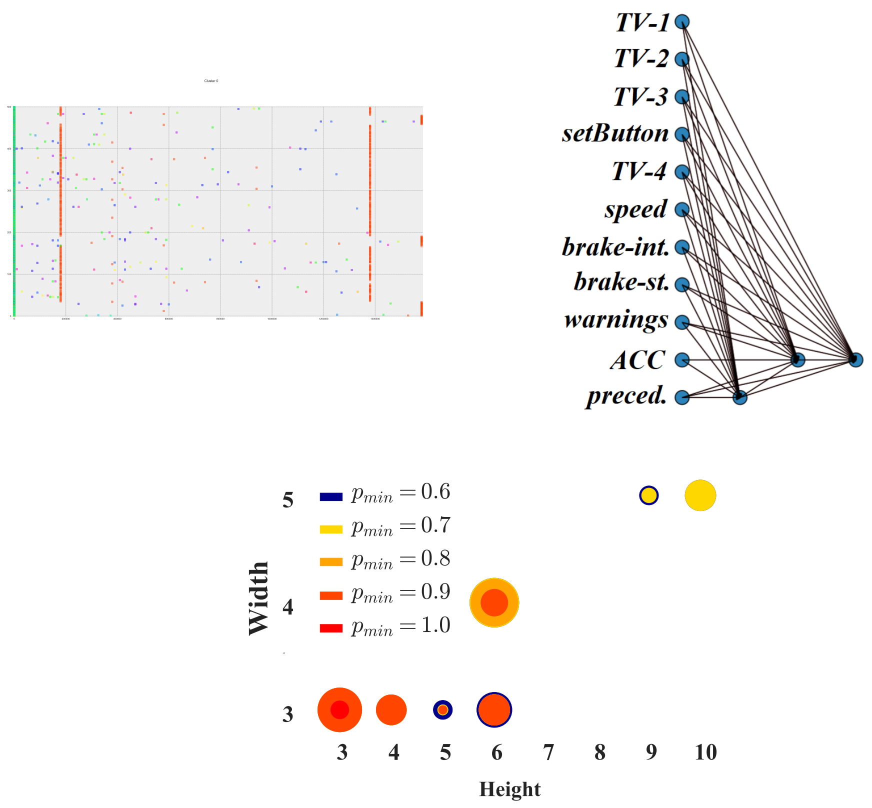


Figure 9.21: Flow of extraction for the ACC data set (TV Cluster 1, Group 0). On the left the MSS segments used for training are given. The right shows the learned structure and the lower part shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

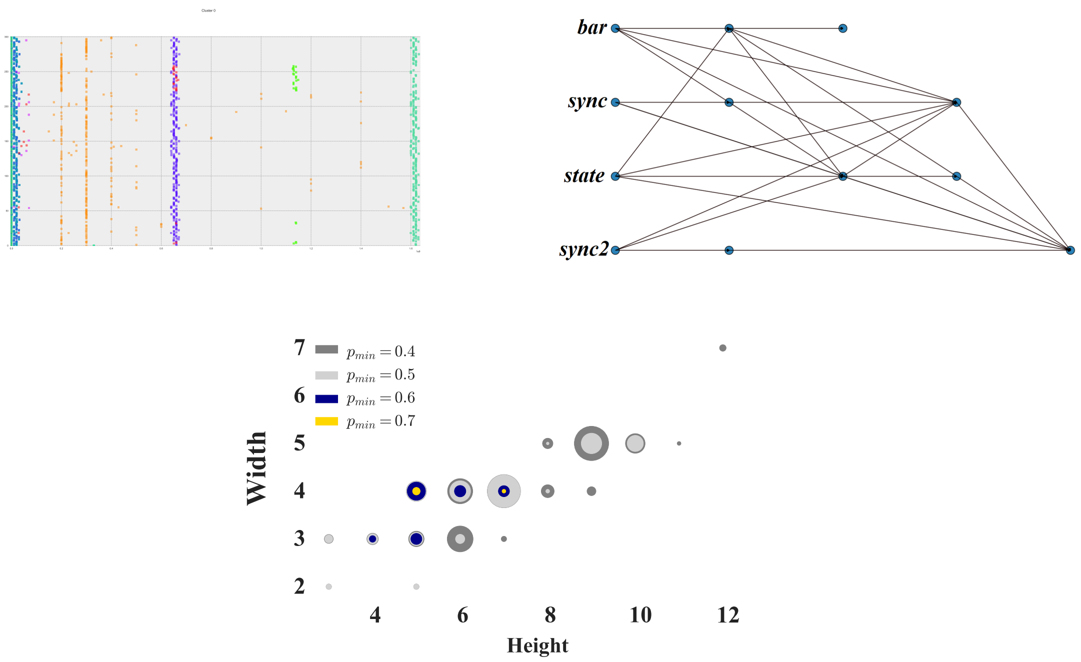


Figure 9.22: Flow of extraction for the Lights data set (TV Cluster 1, Group 0)

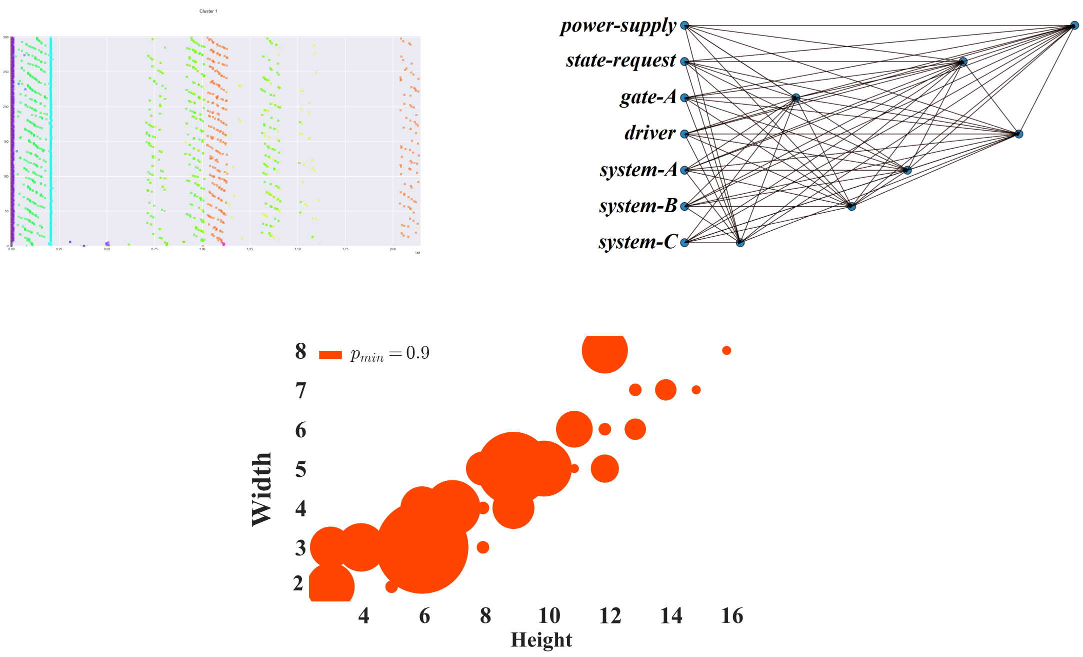


Figure 9.23: Flow of extraction for the Start Up data set (TV Cluster 2, Group 1).

bination count was chosen such that at least 20 % and at most 50 % of the symbols in an expressions are softened.

9.4.5.1 Results of BaySpec

Found Specifications: The numbers of specifications, that were found with BaySpec and a threshold $p_{min} > 0.6$ are 24 for ACC (TV Cluster 1, MSS group 0), 224 for ACC (1, 1) , 104 for Lights (1, 0), 84 for Lights (1, 3), 12 for Lights (2, 3), 53 for Wiper (1, 9), 328 for Wiper (1, 14), 1048 for Start up (2, 1), 36 for Start up (2, 8), 4 for Start up (2, 15), 88 for Start up (3, 0), 209 for Shutdown (1, 0) and 503 for Shutdown (2, 0). The distribution of those specifications in terms of complexity is shown in Figures 9.21, 9.22 and 9.23 for the three examples presented during structure discovery. Further, results are again given in the appendix.
Overall specifications with a high likelihood are supported in the data and thus, correctly learned. Results and main observations for some of the found specifications are discussed in the following.

- **ACC (1, 1):** Here, some non-informative specifications were found, which resemble system states that were present during activation. While those specifications are not wrong they are considered irrelevant. Examples are the state defining that the display has to show or to not show a warning before the state "no preceding car detection". In particular the TV of the warning does not have an influence and is not the cause of an activation. That is,
  G(warnings:none → X(F(preceding:none and X(F(ACC:activated)))))
  G(warnings:showing → X(F(preceding:none and X(F(ACC:activated)))))
  Further, here a main pattern was the preceding car information that occurred in most specifications, which is triggered in two cases. First, if the brake was released, i.e. G(brake:deactivated → X(F(preceding:none and X(F(ACC:activated))))) and second, if certain buttons were pressed, which includes different combinations, including the resume button as
  G(resumeButton:pressed → X(F(preceding:none and X(F(ACC:activated)))))
  and the Set button as
  G(setButton:pressed → X(F(preceding:none and X(F(ACC:activated)))))
  Lastly, various combinations before activation are checked. Those include multiple braking states that are allowed, which are checked similar to the first case.
  G(brake-state:1 → X(F(preceding:none and X(F(ACC:activated)))))
  G(brake-state:3 → X(F(preceding:none and X(F(ACC:activated)))))

- **Lights (1, 0):** Here, three dominant observations are found, which are the activations of the left and right indicator, as well as the synchronization information sent, followed by a deactivation (which has to occur after 3 seconds). The right activation is found as
  G((bar:tip up or state:both off) → X(G(state:right on → X(state:both off)))), the left activation as
  G((bar:tip down or state:both off) → X(G(state:left on → X(state:both off)))) and the synchronization as
  G(sync:indicator continues → X(G(sync:start cycle → X(F(sync:initialize or sync:

indicator continues))))).
Further specifications include the handle bar variation, from pressed to not pressed or the variant of a indicator changing to the defect state.

- **Start up (2, 1):** Here, the dominant specifications are the shutdowns of various system components, where combinations of shutdowns are represented as states in less TVs. That is, one TV might contain multiple variants of system information, which vary little across observations. Thus, the resulting specifications are similar mostly varying at one literal. Further, within this procedure a series of gates are deactivated. An example shutdown procedure of system states is
G((((system-A:a off, b off, c on, d off, e on or power-supply:a) or power-supply:b) or gate-A:hold) $\rightarrow$ X(F(system-A:a off, b off, c off, d off, e off and X(F(driver:not present and X(F(system-B:f off, g on, h off, i off, j on, k off))))))).
Those combinations vary in its initial state nodes, while the consequent procedure is similar in many MSSs. BaySpec captures such intermediate procedures as well, e.g. as
G(power-supply:a $\rightarrow$ X(F(state-request:transmitted)))

Those examples show that the proposed pipeline is able to discover useful specifications in MSSs using BaySpec.

Complexity: In Figures 9.21, 9.22 and 9.23 it can be seen that most specifications are found in Start up, followed by ACC and Lights. Also, Start up produces specifications with highest likelihoods, width and height.
For all sets an increased p_{min} results in less specifications as less paths with sufficiently high likelihood are found. Further, height decreases as less paths are found and less merging is performed.
Moreover, the structure influences the resulting specifications. That is, depending on the number of TVs, its states and the length of each model, the width and height varies. More TVs and states result in a increased width, as after merging more constellations of TVs with states are possible. Longer models yield longer paths, which yields increased height. Such paths also allow for more merging possibilities and thus, an possibly increased width. ACC and Light have a similar dominant width and height area, which is as ACC has more TVs and states with short procedures, while Lights has less TVs with many states and long procedures. In contrast to that, Start up has many TVs, most states and longest procedures yielding the highest complexity among those data sets. Further, in ACC and Lights a threshold of $\chi_{th} = 0.2$ was set for the structure discovery yielding a structure with light minimum correlation, while in Start up the maximum structure is represented. This yields more redundant paths in Start up giving more specifications. This effect is amplified by the fact that Start up has many states that resemble identical behavior in a single TV, yielding results for the same specification with multiple combinations.

Quality: The quality of specifications that are found with BaySpec are discussed in the following.

- *Meaning:* First, with decreasing likelihood of specifications, found specifications may become redundant, non-informative or false positives. First, two types of redundancy might occur. That is, found specifications might be a sub-formula of other specifications or specifications in the final set or might vary in one redundant literal only. Such specifications need to be merged in further post processing.
  Second, specifications might be non-informative (while not wrong). That is, one literal might contain all states of a TV, i.e. it does not matter which state a TV has, making this part of the formula non-informative. The same effect is achieved, if specifications in the result set differ in one literal such that this literal covers all possible states. For instance if the TV warning has two states, the following two formulas might make the first literal irrelevant in terms of its value. First: G(warnings:none → X(F(preceding:none and X(F(ACC:active))))). Second: G(warnings:showing → X(F(preceding:none and X(F(ACC:active))))). In the general case this formula thus, is non-informative. Nevertheless, in the particular case of automotive traces, this formula has a well defined meaning, as it implies that the message of the warnings has to be transmitted before an activation. Another type of non-informative specifications are those that contain no information. Such information is hard to filter based on content as explicit domain knowledge is required. For example in G(detection:traffic in front detected or detection:traffic in back detected) → X(detection:nothing detected), the validity cannot be explicitly tested, as in the general case the state of no detection is legit to never occur, e.g. if a car drives on the motorway for several hours.
  Third, false positive specifications may be found as edge redundancy and estimation errors might lead to paths that were not actually present in the trace. After inspecting the results of above experiments, it was found that the risk of this effect increases for specifications of lower likelihood. In addition to that, false positives might result from the merging operation in BaySpec. For instance in G(bar: tip up → X(F((state:left on or state:Blinker right on) and X(state:both off)))), tipping up should not activate the right indicator, which however, is represented here. Above that, especially for the continuous case formulas might be merged in a way that yields illogical constellations. For example in G((position:d or position:a) → X(G((position:a or position:c) → X(position:b and X(position:a and X((position:b and X(position:c)))))))) it might occur that position a follows position d which is not possible as it needs to pass b and c first.

- *Parallelism:* BaySpec is also able to represent parallel behavior through merging of literals. In the current implementation this parallelism is resembled by connecting multiple possible symbols per literal using an *or* operator. However, this implies that both constellations are allowed. By changing this operator to an *and* operator it can be forced that both states are required together. This problem also occurs in general in found specifications. For instance when merging the set of states of initial TV nodes an *or* allows the consequent procedure if any of the initial states is given. But, often it is rather required that multiple initial states are required at the same time, which are modeled by an *and* operator. This could be a possible extension of BaySpec.

Limitations: First, when applying model checking, with the learned specifications, the data set needs to be filtered to the subset of TVs that are contained in each specification. Else intermediate elements of other TVs could violate the formula.
Further, often the premise found by BaySpec is not sufficient to find the specification within the whole trace, as the formula was learned from one MSS group only (rather than the whole trace). Thus, in the current implementation prior segmentation is required and checking needs to be performed on MSSs that are expected to contain the modeled behavior.
Above that, BaySpec might yield specifications that are valid in a reverse order. For instance G(brake-state:1 $\rightarrow$ X(F(preceding:none and X(F(ACC:deactivated))))) implies that any time a person brakes and no car is preceding the deactivation needs to occur. However, this should only be tested if segments are considered that end with a deactivation. Else this formula is violated anytime that the ACC is not activated. Thus, either prior segmentation or an extended premise needs to be added, that in this example would additionally specify that ACC has to be active.
In addition to that the structure of the trained model defines the TVs that are contained in specifications. Thus, the quality of the discovered structure has a direct influence on the quality of specifications. For instance, in contrast to the Start up case, in the Lights data set less redundant specifications are found, as the Lights data set has a clearer structure with clearer correlations.
Also, it was found that BaySpec performs better for functional procedures with more paths, longer paths and with clearer behavioral correlation (e.g. in Lights), as opposed to short paths with multiple initial states (e.g. in ACC). In the former case likely paths are longer and thus, more meaningful when merged. For the latter case, often multiple states need to be given at the same time to trigger the target state. In the future this can be achieved for instance, by merging resulting specifications with an *and* operator. Alternatively, this can be well achieved with the MPE approach, which is able to capture a snapshot of all allowed states.

Benefits: While, some specifications are redundant, specifications of higher likelihood are well supported by the observations. Hence, the resulting expressions can be well applied on test data. In that case redundant specifications might never be violated and consequently, yields a slight overhead during execution. Nevertheless, this is compensated by the effectiveness of non-redundant expressions, which are able to verify relevant behavior. In practice specifications might be generated automatically and redundant ones filtered if those were not violated within a minimum amount of test data.
Another advantage is the automatic generation of specifications, as manual generation is highly time-consuming. For instance as given in the Start up case, specifications might have multiple states of shape a on, b on, c on, ... per TV, which might contain different constellations. Finding valid transitions among such constellations is easy to automate why it reduces manual effort significantly.

Continuous TVs: BaySpec seems to be applicable to discretized continuous TVs as well. For example, in the Wiper data set rain intensity triggers the wiper engine to start, which is given for various intensities of b, c or d as exemplified in G((position:c or intensity:b) $\rightarrow$ X(F((position:b and X(F(position:a)))))).

In the same way BaySpec was able to identify cyclic specifications of the Wiper moving back and forth e.g. in G((position:d or position:a) → X(G((position:c) → X(position:b and X(position:a and X(position:b and X(position:c))))))), where the first part (premise) is redundant, while the second part contains the cyclic movement to test.

9.4.5.2 Results of MPE

Per data set the MPE approach of Chapter 7 was used, where 100 000 samples are drawn and the 100 most likely results of this MPE method are investigated. The results of applying MPE to the cases above are presented here, while for all remaining cases those are described in the appendix.
For the ACC case both for activation and deactivation multiple constellations of equal likelihood were found. Here, the term likelihood represents the proportional frequency of a certain constellation to be drawn relative to the absolute number of drawn samples. For all other data sets a clear drop in likelihood is present, e.g. from 1 % to 0.3 %. Constellations above this drop showed to mostly represent dominant observed behavior of the data set. Thus, in practice this boundary is used to determine all dominant behaviors.
In the following some examples of found dominant MPE estimates are given.

- **ACC:** For the activation case, two dominating MPEs, that resemble two types of activations, are given in the following. Here, dt ($= \Delta t$) is specified in micro seconds and a subset of TVs is named. Notably, here three nodes for ACC are given as in the training data often a deactivation occurred shortly before activation.

  Example 1: The most likely constellation captures a deactivation through braking, which is followed by an activation. In this case the ACC deactivates shortly and automatically resumes when not disabled by a button.
  ACC - dt=0.0 - activated - dt=0.0 - deactivated - dt=1373978 - activated
  preceding - dt=0.0 - close - dt=20001 - none
  TV-1 - dt=0.0 - state A
  TV-2 - dt=0.0 - state A
  TV-3 - dt=0.0 - state A
  setButton - dt=0.3 - not pressed
  TV-4 - dt=0.0 - state A
  speed - dt=0.2 - a
  brake-intensity - dt=0.0 - d
  brake-state - dt=0.0 - 1
  warnings - dt=0.0 - none

  Example 2: Here an activation is shown, that results from pressing the Set button.
  ACC - dt=0.0 - activated - dt=0.0 - activated - dt=0.0 - activated
  preceding - dt=0.0 - close - dt=0.0 - none
  TV-1 - dt=0.0 - state A
  TV-2 - dt=0.0 - state A
  TV-3 - dt=0.0 - state A

setButton - dt=0.4 - pressed
TV-4 - dt=0.0 - state A
speed - dt=0.0 - b
brake-intensity - dt=0.0 - d
brake-state - dt=0.0 - not pressed
warnings - dt=0.0 - none

- **Lights:** The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named.

  Example 1: The first most likely found structures captures the activation of the right indicator, which turns on and off again. The same is found for the left indicator.
  bar - dt=0.0 - tip up - dt=0.0 - tip up - dt=716707958 - not pressed
  sync - dt=0.0 - indicator continues - dt=22022086 - start cycle - dt=0.0 - initialize
  state - dt=0.0 - both off - dt=15745713 - right on - dt=1740429348 - right off
  sync2 - dt=0.0 - not indicating - dt=17029405 - indicating - dt=1586328954 - not indicating

  Example 2: Here, the defect case is shown.
  bar - dt=0.0 - tip up - dt=0.0 - tip down - dt=555459463 - not pressed
  sync - dt=0.0 - indicator continues - dt=11216975 - start cycle - dt=479977508 - indicator continues
  state - dt=0.0 - both off - dt=16188873 - left on - dt=1749651319 - both off
  sync2 - dt=0.0 - not indicating - dt=3574150 - indicating - dt=640113760 - indication defect

- **Start Up:** The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named. Here the system deactivation is shown.

  Example 1:
  power-supply - dt=0.0 - a - dt=136483490 - d
  state-request - dt=0.0 - none - dt=118302256 - transmitted
  gate-A - dt=0.0 - hold - dt=114115405 - off
  driver - dt=0.0 - offline - dt=5504711 - not present
  system-A - dt=0.0 - a off, b off, c on, d off, e on - dt=21312576 - a off, b off, c on, d off, e off
  system-B - dt=0.0 - f off, g on, h off, i off, j on, k off - dt=0.3 - f off, g on, h off, i off, j off, k off
  system C - dt=0.0 - a off, b off, c off , d off, e on, f on - dt=6327829 - a off, b off, c off , d off, e off, f off

In all three cases the dominant states of the observations were found and thus, yield plausible results. In the first example of the ACC case a deactivation occurs, which is due to braking at time 0. After, 1.3 seconds without any preceding car a reactivation occurs, turning the system back on again. The second case of ACC shows that pressing the Set Button leads to an activation of the ACC. Here, latent state changes are present, i.e. ACC was active throughout the segment. Thus, temporal gaps between events are mostly zero. In the Lights case both examples yield the exact expected behavior. That is, a series of synchronization sequences as well as the right and left indicator activation. Notably, the indicators are both deactivated after some seconds, as here short indication is modeled, as opposed to the long indication which is shown in the appendix. Lastly, in the Start Up example it can be seen that several components are turned off, which is transmitted per system in a single TV. Further, the driver left the car, which indicates that here, the car was closed by the driver from outside of the car in this functional procedure.

Discussion: In comparison to BaySpec MPE allows to include temporal information. Further, specifications that are found with MPE are independent of paths and thus, likely constellations are more likely to represent dominant behavior. Again, here the optimistic choice of edges seems to not influence MPEs performance as it results in mostly plausible constellations.
A drawback of MPE is its lack of direct usability as specification. The reason for this is that it is very specific and hence, does not include strictness information. However, in order to enable this in the future, an approach that is similar to BaySpec might be designed. This method might merge found likely constellations on a model level rather than a path level. Further, in some cases, such as in the ACC case, the first MPE constellations that were drawn in MPE contain very similar or even the same triggering behavior. In ACC among the first 10 entries the brake being at state d causes the analyzed procedure. Those entries varied across redundant variations of other TVs, which have no effect on this procedure. In the ACC case that includes e.g. the speed being at a, b, c or d. However, the intensity or occurrence of this effect depends on the data set. In other data sets, such as the Lights set the two most likely constellations resemble ones right and ones left indication.
Lastly, in case of a target state change (e.g. ACC), another limitation of the MPE approach is that it is not clear what triggered the state (e.g. activation). This is because MPE gives a snapshot of all TVs instead of a relevant subset as in BaySpec. However, it would be more preferable to have an importance score at each node, which could indicate the likelihood of a subset of nodes being the cause of the target state change. This could be solved in the future by merging results of likely MPEs, e.g. in order to identify states that caused a target state change.

9.4.5.3 Conclusion

Both BaySpec and MPE were successfully applied to extract relevant behavior from the investigated functional procedures. BaySpec was able to directly represent specifications as LTL formulas which are used for verification. In particular, along with specifications that are able to verify relevant behavior, BaySpec produces non-informative or redundant specifications, which could be improved in the future. Nevertheless, all resulting LTL formulas can directly be applied on traces that contain similar behavior, i.e. it needs to be ensured that segments under test contain the functional procedure that is to be

tested. This is the case here as the premise that is derived in BaySpec does not include all TVs.
Further, MPE finds a set of dominant states which include temporal information that is represented based on the structure of TSCBNs. With this, likely constellations that are prevalent in the data set are represented. This is especially useful when multiple conditions cause a functional procedure to occur, as it is given in the ACC case, or when better understanding of certain functional procedures is required.

9.5 Conclusion

In this chapter the proposed DM pipeline was used to extract specifications and dominant behavior from five automotive data sets. It was shown that the approach allows to systematically break down complexity of heterogeneous temporal data sets, by preprocessing traces, clustering TVs, clustering and segmenting MSSs, learning TSCBNs to represent functional procedure as well as by applying either BaySpec or MPE to extract the main behavior. This makes systematic automatic extraction of specifications possible.
Above that, an appropriate set of approaches was presented for each stage of the pipeline and its suitability was shown. In addition to that, the influence of individual hyper parameter on the performance of each stage were discussed based on an extensive evaluation.
Moreover, some limitations of those approaches were discussed. In particular this includes extraction of MSSs that vary in behavior. In the given scenario segmentation clustering was targeted to extract a consistent set of MSSs. This raised the problem of computing correlations during structure discovery with TrieDiscover. However, it was shown that by choosing an optimistic number of edges, this problem is omitted yielding meaningful results. In the future either a more diverse set of MSSs needs to be ingested, as is the case in the Lights data set, or a correlation metric needs to be used that computes correlation in terms of steady behavior. This is expected to both reduce complexity of the further processing, as well as the precision of resulting models and specifications. Furthermore, BaySpec works best if multiple paths are given in the structure and thus, performs best if more inter-edges are present across TVs. If this is not the case MPE is preferable to use.
Further, it was shown that by breaking down the complexity into small segments of fewer TVs, TSCBNs are well able to be used to produce specifications. That is, as the DM pipeline produces a high number of models that each yield a set of specifications, rather than mining specifications from the overall trace.
Lastly, in practice, run time performance worsens with higher complexity of the problem. This can however be omitted by choosing the right parameters at each stage. This includes the following. First, during preprocessing less SAX values are chosen, which reduces the number of discretized continuous state changes. Further, after TV clustering smaller sets of TVs are used. In segmentation clustering sub grouping or fusion of segments are used to generate segments of appropriate behavioral variance. Before structure discovery, states of individual TVs might be fused (e.g. different variants of braking), to reduce the number of states per TV. During structure discovery higher correlation thresholds yield less edges, making estimation more efficient. Also, in BaySpec

the minimum probability threshold during path search can be increased, which results in less candidate paths and thus, less computations.

10 Conclusion and Outlook

10.1 Summary

In this work a semi-automated end-to-end pipeline is presented, that allows to extract specifications and dominant behavior from traces of large-scale distributed systems.
First, given a raw trace, a set of relevant dimensions, as well as reduction rules are defined. Those are used within a parallel preprocessing approach to automatically extract a homogeneous trace representation. First, preselection yields a reduced representation that is based on meta-information. This representation is interpreted and reduced using the specified rules, before type-dependent processing is performed on the trace. This ultimately yields a trace in the shape of an interpreted representation of relevant subsets of TVs.
Next, functional procedures need to be identified that are associated with certain domains. This is done in two steps. In the first part, relevant TVs are grouped. This is done based on events that occur in similar intervals and by including expert input. By extracting features for each TV and appropriately reducing those, clustering approaches are used to find correlated clusters. DBSCAN and WaveCluster worked best for the investigated data sets of an automotive trace. In the second part, segments of common functionality are sought. This requires concurrent segmentation and clustering, which in general needs to uncover overlapping functional procedures. Three approaches to solve this were compared, which include a window-based clustering, an extension of LTS and a range segmentation approach. It was found that for non-overlapping procedures those approaches perform equally well. However, LTS and window-based clustering are not well suited to handle functional procedures of varying length, while range segmentation allows for grouping of procedures with flexible lengths. Also, range segmentation showed to be the most scalable among those methods.
The previous steps result in a set of MSSs, which are used to extract specifications. This can be done with existing approaches. However, those exclude structural information, are not well able to handle imperfect traces and cannot find specifications of arbitrary length. Therefore, next, a model is learned as a structured representation of the data which is used as the basis to infer specifications and dominant behavior from it. As MSSs are given, TSCBNs were used for representation. Using TrieDiscover both temporal and multidimensional information is preserved which allows to represent behavioral correlations from the model. If TVs in traces contain sufficient variance in system states, this approach allows to find structures of low complexity. However, if no variance is present an optimistic choice of parameters needs to be used to find structures with redundant edges. That is, as less complex structures tend to miss out potential paths, although those might have been present in actual observations, while in redundant structures BaySpec and MPE are still able to extract dominant behavior. In terms of parameter estimation three approaches EM, VI and MLE-R were compared and show to work

equally well.
Lastly, BaySpec is applied on the resulting structures to extract LTL specifications. Here, found specifications tend to be more meaningful if supported by a higher average likelihood. Less likely specifications get increasingly meaningless or even false. That is, those might form sub-formulas of more likely formulas or contain redundant information. As an alternative approach MPE is used to extract most probable snapshots of the MSSs, which enables to identify dominant behavior. Similarly as in BaySpec here more likely constellations are more likely to represent true behavior that is present in the data, while constellations of lower likelihood might yield false positives.
In a final case study on an automotive data set it was shown that the overall pipeline is consistent and allows to systematically extract specifications. Further, the effect of choosing different hyper parameters at each step is extensively studied. All in all, this allows to use a good parameterization that can be used to apply the designed pipeline for an semi-automated extraction of specifications.

10.2 Lessons Learned

First, with the proposed pipeline it was shown that Specification Mining at large-scale is possible through successive reduction of trace size. This reduction is achieved by extracting functional behavior and by aggregating MSSs into a compact model. In contrast to this, the usage of the raw trace as an input for rule-based or automaton-based mining would yield exploding complexity. However, with the proposed pipeline this is overcome, as it is able to extract multiple branches of TVs and functional segments, that each result in a set of specifications.
By including the expert during preselection, reduction, TV grouping and segmentation, domain relevant functional behavior is found. Doing this in a fully automated manner is currently hardly possible, as it cannot be clearly measured which TVs and MSSs correlate and as meta-data, such as naming of TVs, needs to be considered. Hence, the inclusion of domain knowledge in Specification Mining is essential.
Further, parameterization of the approach depends on the characteristics of the data set. This includes finding an appropriate granularity for groupings of TVs, which is dependent on the behavior that is to be investigated. For instance, height and width of functional procedures does vary based on the distribution of data types in the set. Also, during temporal segmentation different approaches might be required, that depend on the shape of the functional procedures. For instance, this might depend on whether procedures are overlapping or variations of procedure lengths are possible.
Above that, for MSSs with higher variance per state change cbTD with strict correlation thresholds works best. In particular if this variance is low, this threshold needs to be lower, which yields a structure that is determined predominantly based on the order of state changes in the MSSs.
Moreover, an optimal set of input MSSs needs to be found, such that a clear correlation between state changes is present. However, the determination of an appropriate parameterization for segmentation clustering that achieves this is challenging. This is due to the fact, that the question arises, which segment groups might correlate and what variants of segment groups need to be combined.
Lastly, for MSSs with low variance an optimistic number of edges needs to be used in

order to make it applicable for inference of specifications. This is valid, as state changes that occurred consecutively, are connected via conditional dependencies and hence, transitions of actually observed sequences are learned as more likely. With this, inference is possible by focusing on likely constellations only.
The scalability of the current implementation is dependent on the size of the trained model. However, as complexity is broken down by learning multiple small models (as opposed to one big model with all behaviors), with the proposed approach good scalability is achieved, which makes it well usable in practice. Nevertheless, this work is the first step towards scalable Specification Mining, while in the future an extensions to this work might improve performance.

10.3 Answers to Research Questions

1. How does a systematic Specification Mining approach need to be designed to be integrable in current testing and verification procedures of large-scale distributed systems?

   Answer: In such systems many domains are involved and thus, domain specific knowledge needs to be included. That is, the approach needs to be semi-automated. This is given at multiple stages of the proposed DM pipeline including preprocessing, TV clustering and functional segmentation.
   Above that, due to large data sets recorded, a systematic design with multiple steps is required that targets mining towards the analyzing domain, reuses results and reduces redundant computations. This is solved with the pipeline, as each step produces results that are reusable and once parameterized by the expert (in an initial semi-automated run) the overall DM pipeline can be run automatically on similar configurations. The approach can be integrated in an automated design, as the one described in Chapter 3.
   Further, the product is evolving quickly during development, which requires specifications to be adjustable in order to perform well at each iteration of the product. For this, the proposed approach can be both used to design new specifications, as well as to merge new specifications with existing ones such that appropriate strictness of specifications is found. Also, by applying those on many data sets, specifications that are violated too often might be inspected by an expert and removed if deprecated.

2. How can temporal structure be exploited to allow for a more expressive Specification Mining on MSSs?

   Answer: Most existing Specification Mining approaches directly operate on symbols rather than TVs with its states. This excludes inherently present temporal information. In the proposed approach this is overcome by modeling MSSs using a TSCBN, where edges represent only paths that are temporally possible. This allows for optimized representation of both sequentiality and parallelism, which ultimately yields more expressive specifications.

3. How can Specification Mining be performed on noisy and heterogeneous traces in order to produce specifications that compare multiple data types?

   Answer: This is solved by using a preprocessing approach that includes homogenization and reduction. That is, by appropriately quantifying numerical sequences and filtering those for meaningful state changes across all data types, traces are homogenized and noise reduced. In particular removing redundant repeating state changes of discretized numerical values enables comparability of data types.

4. How can domain specific Specification Mining be performed and expert included in the mining procedure?

   Answer: Existing mining approaches do not allow for interactions during specification mining, while this allows for more expressive and relevant specifications. The proposed approach solves this, by including the expert at multiple stages of the pipeline, i.e. during the decisions of which TVs are relevant, how functional procedures are formed and which temporal correlations exist.

5. How can the complexity of a multi-functional large-scale distributed system be broken down, such that effective and efficient mining of relevant specification is enabled?

   Answer: In such systems the system state can be formulated as a set of functional procedures as introduced in Chapter 2. With this assumption it is possible to isolate functional procedures. In the proposed approach it was shown that this isolation can be performed by applying TV clustering and segmentation clustering.

6. How can behavior of functional procedures of MSSs be represented under uncertainty and specifications of arbitrary length extracted?

   Answer: To represent MSSs under uncertainty it is required to represent both the sequential, as well as the parallel behavior of those. This can be done with TSCBNs. The length of specifications is directly associated with the length of the TSCBN, which in turn depends on the length of MSSs used for training. Thus, to achieve specifications of arbitrary length, an appropriate segmentation of MSSs needs to be found. In this work, range segmentation showed, that it can be successfully applied for this task.

7. How can raw communication traces of large-scale distributed systems be processed and functional procedures, as well as specifications identified from those?

   Answer: To enable an end-to-end processing of raw traces, an interpretation approach is required. For massive traces this approach needs to reduce traces early, such that interpretation is performed on relevant subsets only. By then, structuring the interpreted representation as MESs and transforming those to MSSs with the presented approach relevant system states can be extracted.

8. How can functional procedures be identified in high dimensional MSSs of large-scale?

   Answer: In general functional procedures depend on the aspect under analysis, e.g. it might include communication or system state behavior. Therefore, first, identification of those requires to include expert input. Second, from a technical point of view a TV clustering approach is required that can be run on parallelized processing frameworks such as Apace Hadoop. This was presented in Chapter 5. Subsequent segmentation allows to extract functional procedures. The best approach to use for this depends on the composition of functional procedures in a trace (e.g. overlapping vs. non-overlapping).

9. Which combination of approaches is suited to be used at each individual step of the semi-automated processing pipeline?

   Answer: The approaches to use at each step depend on the analysis intent and the shape of the data. For preprocessing a representation needs to be found that results in MSSs of state behavior of the system. In Chapter 4 a raw trace was assumed, that performs early reduction, interpretation and type-dependent reduction to solve this. For TV clustering groups of TVs need to be identified that are correlating, which in the inspected method of Chapter 5 included TVs that are changing in common time spans. This however, excludes TVs that correlate while changing less frequent (e.g. system activations) or missing occurrences that are expected (e.g. no button pressed before activation). In segmentation clustering the type of functional procedure determines the approach to choose. Here range segmentation performed well as active phases in the data were assumed that are sparsely distributed and procedures are of varying lengths. Lastly, for modeling of MSSs TSCBNs in combination with BaySpec and MPE allow to extract meaningful specifications.

10.4 Future Work

Future work includes extension of the pipeline itself, improvement of its steps, variation of its steps, improving its support for other data types and enabling future applications with this approach.

DM pipeline: Future Specification Mining approaches should be able to fully automate extraction and optimization of specifications at a large scale. The current implementation allows for semi-automated mining, which could be improved in the future. That is, artificial intelligence might be used, where an agent creates and optimizes specifications based on a set of rules or based on results from this pipeline. A particular challenge lies in identifying which granularities are reasonable at each step. Above that, the approach could be integrated in an overall testing cycle as it was shown in Chapter 3, where expert feedback that is investigating results of applied specifications, could be used to improve specifications. Moreover, this approach might identify anomalies, when specifications that are automatically learned from one set of MSSs are applied to another set of MSSs that is expected to contain similar behavior.

Improve Steps: The methods that were introduced at each step of the pipeline may be further improved in terms of performance and capabilities. That is, preprocessing of traces might be generalized to ingest and prepare multiple raw data formats. This is important, as in real-life scenarios different devices that record traces of different shape are involved. TV clustering currently groups TVs that occur together, while it might be interesting to include further TVs that are correlated, such as steady dependencies (e.g. device is active). In segmentation clustering the challenge of overlapping segments needs to be solved. Above that, for modeling of MSSs as TSCBNs more efficient and scalable approaches need to be invented that, e.g., might include specialized hardware designs. Lastly, BaySpec and MPE could be extended to improve their limitations which where described in Chapter 8. Those include aspects such as the presence of mandatory or optional parallelism in expressions. Further, both approaches could be combined in the future. MPE yields a set of most probable constellation that each represent a strict specification of the system. By merging such constellations a multidimensional specification could be defined.

Exchange Steps: Another option is to modify the framework by exchanging its steps with other approaches. This is useful as the ultimate goal of the presented pipeline is to be an inherent part of the testing cycle, that systematically learns specifications and is able to improve itself iteratively.
In preprocessing other representations of MSSs might be used, such as one-hot encodings of state changes. Based on this Machine Learning models could be directly trained on the resulting representations. Such models could classify unspecific errors or detect anomalies, e.g., as it was done in [17].
Furthermore, in this work TSCBNs were used for modeling. However, multiple other types of models, such as automata or process models could be used as part of this pipeline in the future. With this, novel inference algorithms could be designed that yield improved extractions of specifications.
Alternatively, existing Specification Mining approaches could be applied directly, which become applicable as data is reduced within the first pipeline steps that are presented in this work.

Applications: The proposed DM pipeline can be used for different application scenarios, that go beyond Specification Mining.
First, it could be extended to allow for improved data understanding. This can be done by highlighting clusters of functional procedures that were found in the trace with the current pipeline. With this, experts can focus on relevant aspects of the trace and investigate interactions between functional procedures.
Further, functional procedures can be used as an indicator of situations, e.g. a driver opening the door.
Above that, behavior among different instances of a system might be investigated by comparing specifications that were learned from different system instance. Then, differences in behavior might indicate optimization potential for one of the two system

instances.
Furthermore, anomaly detection can be performed with this pipeline. This can be achieved by clustering functional procedures such that both correct and abnormal behavior is present. Based on this, models can be used to identify anomalies. For example, Process Mining methods can be used to identify abnormal procedures as branches in a process model. Also, functional procedure distributions could be learned and behavior deviating from it marked as anomaly. Moreover, specifications that were learned from one MSS set, could be applied on another trace under test to reveal discrepancies between both traces as an anomaly. Such anomalies might be used by experts as input to finding similar errors (e.g. by using supervised Machine Learning), to understand the error better or to systematically improve learned specifications based on feedback.

Bibliography

[1] A. S. Tanenbaum and M. Van Steen. *Distributed systems: principles and paradigms.* Prentice-Hall, 2007.

[2] M. Solé, V. Muntés-Mulero, A. I. Rana, and G. Estrada. Survey on models and techniques for root-cause analysis. *arXiv preprint arXiv:1701.08546*, 2017.

[3] A. Fernández, S. del Río, V. López, A. Bawakid, M. J. del Jesus, J. M. Benítez, and F. Herrera. Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(5):380–409, 2014.

[4] A. Mrowca, T. Pramsohler, S. Steinhorst, and U. Baumgarten. Automated interpretation and reduction of in-vehicle network traces at a large scale. In *Proceedings of the 55th Annual Design Automation Conference on - DAC 18.* ACM Press, 2018. URL: `https://doi.org/10.1145%2F3195970.3196000`, `doi:10.1145/3195970.3196000`.

[5] A. Mrowca, B. Moser, and S. Günnemann. Discovering groups of signals in in-vehicle network traces for redundancy detection and functional grouping. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 86–102. Springer, 2018.

[6] C. W. Günther and W. M. van der Aalst. *Mining activity clusters from low-level event logs.* Beta, Research School for Operations Management and Logistics, 2006.

[7] U. Nodelman, C. R. Shelton, and D. Koller. Learning continuous time bayesian networks. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 451–458. Morgan Kaufmann Publishers Inc., 2002.

[8] A. Mrowca, F. Gyrock, and S. Günnemann. Temporal state change bayesian networks for modeling of evolving multivariate state sequences. *Under Review.*, 2020.

[9] C. Baier and J.-P. Katoen. *Principles of model checking.* MIT press, 2008.

[10] A. Mrowca, M. Nocker, S. Steinhorst, and S. Günnemann. Learning temporal specifications from imperfect traces using bayesian inference. In *2019 56th ACM/ESDA/IEEE Design Automation Conference (DAC).* IEEE, 2019.

[11] B. Zhou, J. Too, M. Kulkarni, and S. Bagchi. Wukong: automatically detecting and localizing bugs that manifest at large system scales. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 131–142. ACM, 2013.

[12] P. Wolf, A. Mrowca, T. T. Nguyen, B. Bäker, and S. Günnemann. Pre-ignition detection using deep neural networks: A step towards data-driven automotive diagnostics. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 176–183. IEEE, 2018.

[13] W. E. Wong, V. Debroy, A. Surampudi, H. Kim, and M. F. Siok. Recent catastrophic accidents: Investigating how software was responsible. In *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, pages 14–22. IEEE, 2010.

[14] S. P. Kavulya, K. Joshi, F. Di Giandomenico, and P. Narasimhan. Failure diagnosis of complex systems. In *Resilience assessment and evaluation of computing systems*, pages 239–261. Springer, 2012.

[15] K. Kc and X. Gu. Elt: Efficient log-based troubleshooting system for cloud computing infrastructures. In *2011 IEEE 30th International Symposium on Reliable Distributed Systems*, pages 11–20. IEEE, 2011.

[16] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016.

[17] J.-P. Schulze, A. Mrowca, E. Ren, H.-A. Loeliger, and K. Böttinger. Context by proxy: Identifying contextual anomalies using an output proxy. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2059–2068. ACM, 2019.

[18] A. Hadoop. Hadoop, 2009.

[19] A. V. Mirgorodskiy, N. Maruyama, and B. P. Miller. Problem diagnosis in large-scale computing environments. In *Proc. of SC 2006*, 2006.

[20] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by cluster analysis of execution profiles. In *Proceedings of the 23rd international conference on Software engineering*, pages 339–348. IEEE Computer Society, 2001.

[21] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *OSDI*, volume 4, pages 18–18, 2004.

[22] J. Kwisthout. Most probable explanations in bayesian networks: Complexity and tractability. *International Journal of Approximate Reasoning*, 52(9):1452–1469, 2011.

[23] W. Van der Aalst. Data science in action. In *Process Mining*, pages 3–23. Springer, 2016.

[24] B. F. Van Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, and W. M. Van Der Aalst. The prom framework: A new era in process mining tool support. In *International conference on application and theory of petri nets*, pages 444–454. Springer, 2005.

[25] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. *ACM SIGCOMM Computer Communication Review*, 37(4):13–24, 2007.

[26] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A tool for failure diagnosis in ip networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 173–178. ACM, 2005.

[27] G. Khanna, I. Laguna, F. A. Arshad, and S. Bagchi. Distributed diagnosis of failures in a three tier e-commerce system. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, pages 185–198. IEEE, 2007.

[28] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. Fault localization via risk modeling. *IEEE Transactions on Dependable and Secure Computing*, 7(4):396–409, 2009.

[29] N. Joshi, B. Wilburn, V. Vaish, M. L. Levoy, and M. A. Horowitz. *Automatic color calibration for large camera arrays.* [Department of Computer Science and Engineering], University of California . . . , 2005.

[30] I. Rish, M. Brodie, N. Odintsova, S. Ma, and G. Grabarnik. Real-time problem determination in distributed systems using active probing. In *2004 IEEE/IFIP Network Operations and Management Symposium (IEEE Cat. No. 04CH37507)*, volume 1, pages 133–146. IEEE, 2004.

[31] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial intelligence*, 29(3):241–288, 1986.

[32] W. R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov chain Monte Carlo in practice.* Chapman and Hall/CRC, 1995.

[33] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

[34] R. D. Shachter. Intelligent probabilistic inference, 2013. `arXiv:1304.3446`.

[35] Ö. Sümer, U. A. Acar, A. T. Ihler, and R. R. Mettu. Adaptive exact inference in graphical models. *Journal of Machine Learning Research*, 12(Nov):3147–3186, 2011.

[36] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

[37] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Learning in graphical models*, pages 75–104. Springer, 1998.

[38] K. Kask and R. Dechter. Stochastic local search for bayesian network. In *AISTATS*, 1999.

[39] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[40] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High speed and robust event correlation. *IEEE communications Magazine*, 34(5):82–90, 1996.

[41] L. Monacelli and G. Reali. Evolution of the codebook technique for automatic fault localization. *IEEE Communications Letters*, 15(4):464–466, 2011.

[42] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, and C. Jard. Fault detection and diagnosis in distributed systems: an approach by partially stochastic petri nets. *Discrete event dynamic systems*, 8(2):203–231, 1998.

[43] Q. Luo, A. Nair, M. Grechanik, and D. Poshyvanyk. Forepost: Finding performance problems automatically with feedback-directed learning software testing. *Empirical Software Engineering*, 22(1):6–56, 2017.

[44] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 36–43. IEEE, 2004.

[45] Y.-Y. M. Chen, A. J. Accardi, E. Kiciman, D. A. Patterson, A. Fox, and E. A. Brewer. *Path-based failure and evolution management.* University of California, Berkeley, 2004.

[46] X. Xu, L. Zhu, I. Weber, L. Bass, and D. Sun. Pod-diagnosis: Error diagnosis of sporadic operations on cloud applications. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 252–263. IEEE, 2014.

[47] R. J. Kate and R. J. Mooney. Rj: Probabilistic abduction using markov logic networks. In *In: IJCAI-09 Workshop on Plan, Activity, and Intent Recognition.* Citeseer, 2009.

[48] P. Singla and R. J. Mooney. Abductive markov logic for plan recognition. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[49] G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, pages 2178–2185. AAAI Press/International Joint Conferences on Artificial Intelligence; Menlo . . . , 2011.

[50] S. M. Kazemi and D. Poole. Knowledge compilation for lifted probabilistic inference: Compiling to a low-level language. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2016.

[51] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma. Automated known problem diagnosis with event traces. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 375–388. ACM, 2006.

[52] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proc. of DSN 2002*, 2002.

[53] Z. Lan, Z. Zheng, and Y. Li. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(2):174–187, 2009.

[54] L. Mariani and F. Pastore. Automated identification of failure causes in system logs. In *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, pages 117–126. IEEE, 2008.

[55] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 105–118. ACM, 2005.

[56] A. Babenko, L. Mariani, and F. Pastore. Ava: automated interpretation of dynamically detected anomalies. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 237–248. Citeseer, 2009.

[57] R. Ren, X. Fu, J. Zhan, and W. Zhou. Logmaster: Mining event correlations in logs of large scale cluster systems. *arXiv preprint arXiv:1003.0951*, 2010.

[58] G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. *ACM Sigplan Notices*, 37(1):4–16, 2002.

[59] A. Wasylkowski and A. Zeller. Mining temporal specifications from object usage. *Automated Software Engineering*, 18(3-4):263–292, 2011. `doi:10.1007/s10515-011-0084-1`.

[60] M. K. Ramanathan, A. Grama, and S. Jagannathan. Static specification inference using predicate mining. In *ACM SIGPLAN Notices*, volume 42, pages 123–134. ACM, 2007.

[61] S. Shoham, E. Yahav, S. J. Fink, and M. Pistoia. Static specification mining using automata-based abstractions. *IEEE Transactions on Software Engineering*, 34(5):651–666, 2008. `doi:10.1109/TSE.2008.63`.

[62] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as deviant behavior: A general approach to inferring errors in systems code. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 57–72. ACM, 2001.

[63] R. Alur, P. Černỳ, P. Madhusudan, and W. Nam. Synthesis of interface specifications for java classes. *ACM SIGPLAN Notices*, 40(1):98–109, 2005. `doi:10.1145/1047659.1040314`.

[64] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 267–277. ACM, 2011.

[65] D. Lo and S.-C. Khoo. Smartic: towards building an accurate, robust and scalable specification miner. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 265–275. ACM, 2006.

[66] M. Gabel and Z. Su. Javert: fully automatic mining of general temporal properties from dynamic traces. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 339–349. ACM, 2008.

[67] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE transactions on software engineering*, 27(2):99–123, 2001.

[68] C. Lemieux, D. Park, and I. Beschastnikh. General ltl specification mining (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 81–92. IEEE, 2015.

[69] D. Lo, S.-C. Khoo, and C. Liu. Mining temporal rules for software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, 20(4):227–247, 2008.

[70] G. Cutulenco, Y. Joshi, A. Narayan, and S. Fischmeister. Mining timed regular expressions from system traces. In *Proceedings of the 5th International Workshop on Software Mining*, pages 3–10. ACM, 2016.

[71] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: mining temporal api rules from imperfect traces. In *Proceedings of the 28th international conference on Software engineering*, pages 282–291. ACM, 2006.

[72] W. Li, A. Forin, and S. A. Seshia. Scalable specification mining for verification and diagnosis. In *Proceedings of the 47th design automation conference*, pages 755–760. ACM, 2010.

[73] M. P. Robillard, E. Bodden, D. Kawrykow, M. Mezini, and T. Ratchford. Automated api property inference techniques. *IEEE Transactions on Software Engineering*, 39(5):613–637, 2013.

[74] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson. Goldmine: Automatic assertion generation using data mining and static analysis. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 626–629. IEEE, 2010.

[75] M. Bonato, G. Di Guglielmo, M. Fujita, F. Fummi, and G. Pravadelli. Dynamic property mining for embedded software. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 187–196. ACM, 2012.

[76] A. Danese, T. Ghasempouri, and G. Pravadelli. Automatic extraction of assertions from execution traces of behavioural models. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 67–72. IEEE, 2015.

[77] Y. Zhang, G. Gantt, M. Rychlinski, R. Edwards, J. Correia, and C. Wolf. Connected vehicle diagnostics and prognostics, concept, and initial practice. *IEEE Trans. on Reliability*, 58(2), 2009.

[78] R. Prytz, S. Nowaczyk, and S. Byttner. Towards relation discovery for diagnostics. In *Proceedings of the First International Workshop on Data Mining for Service and Maintenance - KDD4Service 11*. ACM Press, 2011. URL: `https://doi.org/10.1145%2F2018673.2018678`, `doi:10.1145/2018673.2018678`.

[79] S. Nowaczyk, R. Prytz, and S. Byttner. Ideas for fault detection using relation discovery. In *The 27th annual workshop of the Swedish Artificial Intelligence Society (SAIS); 14-15 May 2012; Örebro; Sweden*, number 071, pages 1–6. Linköping University Electronic Press, 2012.

[80] I. A. Raptis, C. Sconyers, R. Martin, R. Mah, N. Oza, D. Mavris, and G. J. Vachtsevanos. A particle filtering-based framework for real-time fault diagnosis of autonomous vehicles. In *Annual Conference of the Prognostics and Health Management Society*, 2013.

[81] P. Taylor, F. Adamu-Fika, S. S. Anand, A. Dunoyer, N. Griffiths, and T. Popham. Road type classification through data mining. In *Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications - AutomotiveUI 12*. ACM Press, 2012. URL: `https://doi.org/10.1145%2F2390256.2390295`, `doi:10.1145/2390256.2390295`.

[82] P. Taylor, N. Griffiths, A. Bhalerao, A. Dunoyer, T. Popham, and Z. Xu. Feature selection in highly redundant signal data: A case study in vehicle telemetry data and driver monitoring. In *International Workshop Autonomous Intelligent Systems: Multi-Agents and Data Mining. Springer*, pages 25–36. Citeseer, 2013.

[83] A. Sathyanarayana, S. Nageswaren, H. Ghasemzadeh, R. Jafari, and J. H. Hansen. Body sensor networks for driver distraction identification. In *Vehicular Electronics and Safety, 2008. ICVES 2008. IEEE International Conference on*, pages 120–125. IEEE, 2008.

[84] H. Guo, J. A. Crossman, Y. L. Murphey, and M. Coleman. Automotive signal diagnostics using wavelets and machine learning. *IEEE transactions on vehicular technology*, 49(5):1650–1662, 2000.

[85] J. A. Crossman, H. Guo, Y. L. Murphey, and J. Cardillo. Automotive signal fault diagnostics-part i: signal fault analysis, signal segmentation, feature extraction and quasi-optimal feature selection. *IEEE Transactions on Vehicular Technology*, 52(4):1063–1075, 2003.

[86] P. Agarwal, G. Shroff, S. Saikia, and Z. Khan. Efficiently discovering frequent motifs in large-scale sensor data. In *Proc. IKDD CODS 2015*, 2015.

[87] R. Kruse, M. Steinbrecher, and C. Moewes. Data mining applications in the automotive industry. In *4th International Workshop on Reliable Engineering Computing*, 2010.

[88] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery - DMKD 03*. ACM Press, 2003. URL: `https://doi.org/10.1145%2F882082.882086`, `doi:10.1145/882082.882086`.

[89] R. Prytz, S. Nowaczyk, T. Rögnvaldsson, and S. Byttner. Predicting the need for vehicle compressor repairs using maintenance records and logged vehicle data. *Engineering applications of artificial intelligence*, 41:139–150, 2015.

[90] E. Frisk, M. Krysander, and E. Larsson. Data-driven lead-acid battery prognostics using random survival forests. In *Proceedings of the annual conference of the prognostics and health management society. Fort Worth, Texas, USA*, 2014.

[91] J. Wang. Constraint-based event trace reduction. In *Proc. of SIGSOFT 2016*, pages 1106–1108. ACM, 2016.

[92] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman. Temporal event sequence simplification. *IEEE Trans. on visualization and computer graphics*, 19(12):2227–2236, 2013.

[93] Y. Li and L. Guo. An efficient network anomaly detection scheme based on tcm-knn algorithm and data reduction mechanism. In *IAW 2007*, pages 221–227. IEEE, 2007.

[94] J. J. Davis and A. J. Clark. Data preprocessing for anomaly based network intrusion detection: A review. *Computers & Security*, 30(6):353–375, 2011.

[95] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proc. of ICDM 2001*, 2001.

[96] C. GmbH. Carmen analyzer. *Web page: https://codemanufaktur.com/projekte/bmw- car-measurement-environment/*, last modified: 2017.

[97] A. de Haro-García and N. García-Pedrajas. A divide-and-conquer recursive approach for scaling up instance selection algorithms. *Data Mining and Knowledge Discovery*, 18(3):392–418, 2009.

[98] B. Schlegel and B. Sick. Design and optimization of an autonomous feature selection pipeline for high dimensional, heterogeneous feature spaces. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–9. IEEE, 2016.

[99] I.-H. Chung, R. E. Walkup, H.-F. Wen, and H. Yu. Mpi performance analysis tools on blue gene/l. In *Proc. of SC 2006*, pages 16–16. ACM/IEEE, 2006.

[100] K. Mohror and K. L. Karavanic. Evaluating similarity-based trace reduction techniques for scalable performance analysis. In *Proc. of SC 2009*, page 55. ACM, 2009.

[101] L. Carrington, A. Snavely, X. Gao, and N. Wolter. A performance prediction framework for scientific applications. *Proc. of ICCS 2003*, pages 701–701, 2003.

[102] G. Aguilera, P. J. Teller, M. Taufer, and F. Wolf. A systematic multi-step methodology for performance analysis of communication traces of distributed applications based on hierarchical clustering. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8–pp. IEEE, 2006.

[103] B. Mihajlović, Ž. Žilić, and W. J. Gross. Architecture-aware real-time compression of execution traces. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(4):75, 2015.

[104] I. Batal, G. F. Cooper, D. Fradkin, J. Harrison Jr, F. Moerchen, and M. Hauskrecht. An efficient pattern mining approach for event detection in multivariate temporal data. *Knowledge and information systems*, 46(1):115–150, 2016.

[105] P. Chaovalit, A. Gangopadhyay, G. Karabatis, and Z. Chen. Discrete wavelet transform-based time series analysis and mining. *ACM Computing Surveys (CSUR)*, 43(2):6, 2011.

[106] T.-c. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[107] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[108] C. S. Möller-Levet, F. Klawonn, K.-H. Cho, and O. Wolkenhauer. Fuzzy clustering of short time-series and unevenly distributed sampling points. In *Advances in Intelligent Data Analysis V*, pages 330–340. Springer Berlin Heidelberg, 2003. URL: `https://doi.org/10.1007%2F978-3-540-45231-7_31`, `doi:10.1007/978-3-540-45231-7_31`.

[109] A. Nanopoulos, R. Alcock, and Y. Manolopoulos. Feature-based classification of time-series data. *International Journal of Computer Research*, 10(3):49–61, 2001.

[110] X. Wang, K. A. Smith, and R. J. Hyndman. Dimension reduction for clustering time series using global characteristics. In *Lecture Notes in Computer Science*, pages 792–795. Springer Berlin Heidelberg, 2005. URL: `https://doi.org/10.1007%2F11428862_108`, `doi:10.1007/11428862_108`.

[111] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series. In *In proc. workshop on clustering high dimensionality data and its applications*. Citeseer, 2003.

[112] S. Lee, D. Kwon, and S. Lee. Dimensionality reduction for indexing time series based on the minimum distance. *Journal of Information Science and Engineering*, 19(4):697–711, 2003.

[113] E. Keogh, S. Chu, D. Hart, and M. Pazzani. Segmenting time series: A survey and novel approach. In *Data mining in time series databases*, pages 1–21. World Scientific, 2004.

[114] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *VLDB*, volume 385, page 99, 2000.

[115] T. Räsänen and M. Kolehmainen. Feature-based clustering for electricity use time series data. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 401–412. Springer, 2009.

[116] E. Keogh. Fast similarity search in the presence of longitudinal scaling in time series databases. In *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence*, pages 578–584. IEEE, 1997.

[117] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *KDD*, volume 98, pages 16–22, 1998.

[118] T.-c. Fu, F.-l. Chung, V. Ng, and R. Luk. Pattern discovery from stock time series using self-organizing maps. In *Workshop Notes of KDD2001 Workshop on Temporal Data Mining*, pages 26–29, 2001.

[119] J. Catlett. On changing continuous attributes into ordered discrete attributes. In *European working session on learning*, pages 164–178. Springer, 1991.

[120] L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. In *International conference on pervasive computing*, pages 1–17. Springer, 2004.

[121] S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[122] L. Kaufman, P. Rousseeuw, and Y. Dodge. Clustering by means of medoids in statistical data analysis based on the, 1987.

[123] T. A. Runkler. Data analytics. *Wiesbaden: Springer. doi*, 10:978–3, 2012.

[124] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[125] W. Wang, J. Yang, R. Muntz, et al. Sting: A statistical information grid approach to spatial data mining. In *VLDB*, volume 97, pages 186–195, 1997.

[126] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[127] T. Kohonen. Clustering, taxonomy, and topological maps of patterns. In *Proc. of 6th Int'l. Conf. Pattern Recognition IEEE Computer Society Press*, pages 114–128. Silver Spring, 1988.

[128] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514, 2018.

[129] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3861–3870. JMLR. org, 2017.

[130] P. Huang, Y. Huang, W. Wang, and L. Wang. Deep embedding network for clustering. In *2014 22nd International Conference on Pattern Recognition*, pages 1532–1537. IEEE, 2014.

[131] S. A. Shah and V. Koltun. Deep continuous clustering. *arXiv preprint arXiv:1803.01449*, 2018.

[132] J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487, 2016.

[133] F. Li, H. Qiao, and B. Zhang. Discriminatively boosted image clustering with fully convolutional auto-encoders. *Pattern Recognition*, 83:161–173, 2018.

[134] M. P. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197, 2010.

[135] J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

[136] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[137] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

[138] N. S. Madiraju, S. M. Sadat, D. Fisher, and H. Karimabadi. Deep temporal clustering: Fully unsupervised learning of time-domain features. *arXiv preprint arXiv:1802.01059*, 2018.

[139] C. Shaw and G. King. Using cluster analysis to classify time series. *Physica D: Nonlinear Phenomena*, 58(1-4):288–298, 1992.

[140] A. J. Lee, M.-C. Lin, R.-T. Kao, and K.-T. Chen. An effective clustering approach to stock market prediction. In *PACIS*, page 54, 2010.

[141] C. Goutte, P. Toft, E. Rostrup, F. Å. Nielsen, and L. K. Hansen. On clustering fmri time series. *NeuroImage*, 9(3):298–310, 1999.

[142] A. Debrégeas and G. Hébrail. Interactive interpretation of kohonen maps applied to curves. In *KDD*, volume 1998, pages 179–183, 1998.

[143] Y. L. Murphey, M. Masrur, Z. Chen, and B. Zhang. Model-based fault diagnosis in electric drives using machine learning. *IEEE/ASME Transactions on Mechatronics*, 11(3):290–303, jun 2006. URL: `https://doi.org/10.1109%2Ftmech.2006.875568`, `doi:10.1109/tmech.2006.875568`.

[144] K. Kim and A. Parlos. Induction motor fault diagnosis based on neuropredictors and wavelet signal processing. *IEEE/ASME Transactions on Mechatronics*, 7(2):201–219, jun 2002. URL: `https://doi.org/10.1109%2Ftmech.2002.1011258`, `doi:10.1109/tmech.2002.1011258`.

[145] M. E. Orchard. *A particle filtering-based framework for on-line fault diagnosis and failure prognosis*. PhD thesis, Georgia Institute of Technology, 2007.

[146] S. Voronov, D. Jung, and E. Frisk. Heavy-duty truck battery failure prognostics using random survival forests. *IFAC-PapersOnLine*, 49(11):562–569, 2016.

[147] H. Zheng, H. Zhang, H. Meng, and X. Wang. Qualitative modeling of vehicle behavior for scenario parsing. In *2006 IEEE Intelligent Transportation Systems Conference*. IEEE, 2006. URL: `https://doi.org/10.1109%2Fitsc.2006.1706813`, `doi:10.1109/itsc.2006.1706813`.

[148] P. Taylor, N. Griffths, A. Bhalerao, T. Popham, X. Zhou, and A. Dunoyer. Redundant feature selection for telemetry data. In *International Workshop on Agents and Data Mining Interaction*, pages 53–65. Springer, 2013.

[149] P. Chaovalit. *Clustering transient data streams by example and by variable*. PhD thesis, University of Maryland, Baltimore County, 2009.

[150] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of intelligent information systems*, 17(2-3):107–145, 2001.

[151] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[152] T. K. Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[153] J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[154] P. Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[155] P. J. Rousseeuw and L. Kaufman. Finding groups in data. *Hoboken: Wiley Online Library*, 1990.

[156] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

[157] B. Horn, B. Klaus, and P. Horn. *Robot vision.* MIT press, 1986.

[158] D. G. Roussinov and H. Chen. A scalable self-organizing map algorithm for textual classification: A neural network approach to thesaurus generation. 1998.

[159] S. Beniwal and J. Arora. Classification and feature selection techniques in data mining. *International Journal of Engineering Research & Technology*, 1(6):1–6, 2012.

[160] M. Dash and H. Liu. Feature selection for classification. *Intelligent data analysis*, 1(1-4):131–156, 1997.

[161] G. W. Milligan. A monte carlo study of thirty internal criterion measures for cluster analysis. *Psychometrika*, 46(2):187–199, 1981.

[162] G. W. Milligan and M. C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.

[163] L. Vendramin, R. J. Campello, and E. R. Hruschka. Relative clustering validity criteria: A comparative overview. *Statistical analysis and data mining: the ASA data science journal*, 3(4):209–235, 2010.

[164] A. Zimmermann. Understanding episode mining techniques: Benchmarking on diverse, realistic, artificial data. *Intelligent Data Analysis*, 18(5):761–791, 2014.

[165] J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In *ACM SIGMOD Record*, volume 23, pages 115–125. ACM, 1994.

[166] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[167] X. Yan, J. Han, and R. Afshar. Clospan: Mining: Closed sequential patterns in large datasets. In *Proceedings of the 2003 SIAM international conference on data mining*, pages 166–177. SIAM, 2003.

[168] P. T. X. Y. J. Han, P. Tzvetkov, and X. Yan. Tsp: Mining top-k closed sequential patterns. *National Science Foundation under Grant*, (02-09199), 2003.

[169] G. Casas-Garriga. Summarizing sequential data with closed partial orders. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 380–391. SIAM, 2005.

[170] M. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. *IEEE Transactions on knowledge and data engineering*, 14(3):530–552, 2002.

[171] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3):259–289, 1997.

[172] S. Laxman, P. Sastry, and K. Unnikrishnan. A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 410–419. ACM, 2007.

[173] T. Calders, N. Dexters, and B. Goethals. Mining frequent itemsets in a stream. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 83–92. IEEE, 2007.

[174] N. Tatti. Significance of episodes based on minimal windows. In *2009 Ninth IEEE International Conference on Data Mining*, pages 513–522. IEEE, 2009.

[175] R. Gwadera, M. J. Atallah, and W. Szpankowski. Reliable detection of episodes in event sequences. *Knowledge and Information Systems*, 7(4):415–437, 2005.

[176] R. Gwadera, M. Atallah, and W. Szpankowski. Markov models for identification of significant episodes. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 404–414. SIAM, 2005.

[177] D. Patel, W. Hsu, and M. L. Lee. Mining relationships among interval-based events for classification. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 393–404. ACM, 2008.

[178] K.-Y. Chen, B. P. Jaysawal, J.-W. Huang, and Y.-B. Wu. Mining frequent time interval-based event with duration patterns from temporal database. In *2014 international conference on data science and advanced analytics (DSAA)*, pages 548–554. IEEE, 2014.

[179] Y.-C. Chen, J.-C. Jiang, W.-C. Peng, and S.-Y. Lee. An efficient algorithm for mining time interval-based patterns in large database. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 49–58. ACM, 2010.

[180] N. Méger and C. Rigotti. Constraint-based mining of episode rules and optimal window sizes. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 313–324. Springer, 2004.

[181] G. Shani, C. Meek, and A. Gunawardana. Hierarchical probabilistic segmentation of discrete events. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 974–979. IEEE, 2009.

[182] D. Graves and W. Pedrycz. Multivariate segmentation of time series with differential evolution. In *IFSA/EUSFLAT Conf.*, pages 1108–1113. Citeseer, 2009.

[183] P. Cohen, N. Adams, and B. Heeringa. Voting experts: An unsupervised algorithm for segmenting sequences. *Intelligent Data Analysis*, 11(6):607–625, 2007.

[184] M. Ramoni, P. Sebastiani, and P. Cohen. Bayesian clustering by dynamics. *Machine learning*, 47(1):91–121, 2002.

[185] D. Hallac, S. Vare, S. Boyd, and J. Leskovec. Toeplitz inverse covariance-based clustering of multivariate time series data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 215–223. ACM, 2017.

[186] C. W. Günther, A. Rozinat, and W. M. Van Der Aalst. Activity mining by global trace segmentation. In *International Conference on Business Process Management*, pages 128–139. Springer, 2009.

[187] D. R. Ferreira, F. Szimanski, and C. G. Ralha. A hierarchical markov model to understand the behaviour of agents in business processes. In *International Conference on Business Process Management*, pages 150–161. Springer, 2012.

[188] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[189] V. Niennattrakul and C. A. Ratanamahatana. On clustering multimedia time series data using k-means and dynamic time warping. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE07)*. IEEE, 2007. URL: `https://doi.org/10.1109%2Fmue.2007.165`, `doi:10.1109/mue.2007.165`.

[190] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *icccn*, page 0215. IEEE, 2001.

[191] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[192] T. Dean et al. A model for reasoning about persistence and causation. *Computational intelligence*, 5(2), 1989.

[193] U. Nodelman et al. Continuous time bayesian networks. In *Proc. of UAI*. Morgan Kaufmann Publishers Inc., 2002.

[194] U. Kjærulff. dhugin: A computational system for dynamic time-sliced bayesian networks. *International journal of forecasting*, 11(1), 1995.

[195] K. G. Olesen et al. Diagnosing lyme disease-tailoring patient specific bayesian networks for temporal reasoning. In *Probabilistic Graphical Models*, 2006.

[196] H. Boudali et al. A discrete-time bayesian network reliability modeling and analysis framework. *Reliability Engineering & System Safety*, 87(3), 2005.

[197] E. Santos Jr et al. Probabilistic temporal networks: A unified framework for reasoning with time and uncertainty. *International Journal of Approximate Reasoning*, 20(3), 1999.

[198] A. Tucker. *The Automatic Explanation of Multivariate Time Series*. PhD thesis, Ph. D. thesis, Birkbeck College, University of London, United Kingdom, 2001.

[199] L. Song, M. Kolar, and E. P. Xing. Time-varying dynamic bayesian networks. In *Advances in neural information processing systems*, pages 1732–1740, 2009.

[200] G. Arroyo-Figueroa et al. A temporal bayesian network for diagnosis and prediction. In *Proc. of UAI*. Morgan Kaufmann Publishers Inc., 1999.

[201] S. F. Galán et al. Modeling dynamic causal interaction with bayesian networks: temporal noisy gates. In *Proc. 2nd Inter. Workshop on Causal Networks*, 2000.

[202] S. F. Galán et al. Networks of probabilistic events in discrete time. *International Journal of Approximate Reasoning*, 30(3):181–202, 2002.

[203] G. Arroyo-Figueroa et al. Temporal bayesian network of events for diagnosis and prediction in dynamic domains. *Applied Intelligence*, 23(2), 2005.

[204] C. F. Aliferis et al. A structurally and temporally extended bayesian belief network model: definitions, properties, and modeling techniques. In *Proc. of UAI*. Morgan Kaufmann Publishers Inc., 1996.

[205] W. Y. Kwon and I. H. Suh. A temporal bayesian network with application to design of a proactive robotic assistant. In *2012 IEEE International Conference on Robotics and Automation*, pages 3685–3690. IEEE, 2012.

[206] A. Y. Tawfik et al. Temporal reasoning and bayesian networks. *Computational Intelligence*, 16(3), 2000.

[207] C. Berzuini. Representing time in causal probabilistic networks. In *Machine Intelligence and Pattern Recognition*, volume 10. Elsevier, 1990.

[208] V. Ryabov et al. Probabilistic temporal interval networks. In *Proc. of TIME 2004*. IEEE, 2004.

[209] P. Weber and L. Jouffe. Complex system reliability modelling with dynamic object oriented bayesian networks (doobn). *Reliability Engineering & System Safety*, 91(2):149–162, 2006.

[210] F. De Carlo, O. Borgia, and M. Tucci. Imperfect maintenance modelling by dynamic object oriented bayesian networks. *International Journal of Engineering and Technology*, 5(5):4282–4295, 2013.

[211] M. Bouissou et al. A new formalism that combines advantages of fault-trees and markov models: Boolean logic driven markov processes. *Reliability Engineering & System Safety*, 82(2), 2003.

[212] J. B. Dugan et al. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on reliability*, 41(3), 1992.

[213] S. Swaminathan et a. The event sequence diagram framework for dynamic probabilistic risk assessment. *Reliability Engineering & System Safety*, 63(1), 1999.

[214] Y. Gu, Y. Sun, and J. Gao. The co-evolution model for social network evolving and opinion migration. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 175–184. ACM, 2017.

[215] M. Corneli, P. Latouche, and F. Rossi. Modelling time evolving interactions in networks through a non stationary extension of stochastic block models. In *Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference on*, pages 1590–1591. IEEE, 2015.

[216] G. Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[217] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[218] G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

[219] M. Bartlett and J. Cussens. Advances in bayesian network learning using integer programming. *arXiv preprint arXiv:1309.6825*, 2013.

[220] C. Yuan and B. Malone. Learning optimal bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–65, 2013.

[221] P. Van Beek and H.-F. Hoffmann. Machine learning of bayesian networks using constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 429–445. Springer, 2015.

[222] A. Tucker, X. Liu, and A. Ogden-Swift. Evolutionary learning of dynamic probabilistic models with large time lags. *International Journal of Intelligent Systems*, 16(5):621–645, 2001.

[223] L. M. de Campos and J. F. Huete. Approximating causal orderings for bayesian networks using genetic algorithms and simulated annealing. In *Proceedings of the Eighth IPMU Conference*, volume 1, pages 333–340, 2000.

[224] H. Xing-Chen, Q. Zheng, T. Lei, and S. Li-Ping. Learning bayesian network structures with discrete particle swarm optimization algorithm. In *2007 IEEE Symposium on Foundations of Computational Intelligence*, pages 47–52. IEEE, 2007.

[225] P. Spirtes, C. Glymour, and R. Scheines. Causation, prediction, and search. adaptive computation and machine learning, 2000.

[226] P. Spirtes and C. Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72, 1991.

[227] L. M. De Campos, J. M. Fernández-Luna, and J. M. Puerta. An iterated local search algorithm for learning bayesian networks with restarts based on conditional independence tests. *International Journal of Intelligent Systems*, 18(2):221–235, 2003.

[228] X. Fan, B. M. Malone, and C. Yuan. Finding optimal bayesian network structures with constraints learned from data. In *UAI*, pages 200–209, 2014.

[229] M. Scanagatta, C. P. de Campos, G. Corani, and M. Zaffalon. Learning bayesian networks with thousands of variables. In *Advances in neural information processing systems*, pages 1864–1872, 2015.

[230] R. Daly, Q. Shen, and S. Aitken. Learning bayesian networks: approaches and issues. *The knowledge engineering review*, 26(2):99–157, 2011.

[231] M. Singh and M. Valtorta. Construction of bayesian network structures from data: a brief survey and an efficient algorithm. *International journal of approximate reasoning*, 12(2):111–131, 1995.

[232] D. Dash and M. J. Druzdzel. A hybrid anytime algorithm for the construction of causal models from sparse data. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 142–149. Morgan Kaufmann Publishers Inc., 1999.

[233] I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 65(1):31–78, 2006.

[234] K. P. Murphy and S. Russell. Dynamic bayesian networks: representation, inference and learning. 2002.

[235] J. W. Robinson and A. J. Hartemink. Learning non-stationary dynamic bayesian networks. *Journal of Machine Learning Research*, 11(Dec):3647–3680, 2010.

[236] T. Savickas and O. Vasilecas. Business process event log transformation into bayesian belief network. 2014.

[237] T. Savickas and O. Vasilecas. Decision support using belief network constructed from business process event log. *Informatica*, 28(4):687–701, 2017.

[238] R. A. Sutrisnowati, H. Bae, and M. Song. Bayesian network construction from event log for lateness analysis in port logistics. *Computers & Industrial Engineering*, 89:53–66, 2015.

[239] R. A. Sutrisnowati, H. Bae, J. Park, and B.-H. Ha. Learning bayesian network from event logs using mutual information test. In *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, pages 356–360. IEEE, 2013.

[240] W. Van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[241] A. Weijters, W. M. van Der Aalst, and A. A. De Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166:1–34, 2006.

[242] S. J. Leemans, D. Fahland, and W. M. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *International conference on business process management*, pages 66–78. Springer, 2013.

[243] J. Bubenzer. Minimization of acyclic dfas. In *Stringology*, pages 132–146, 2011.

[244] D. Barber. *Bayesian reasoning and machine learning.* Cambridge University Press, 2012.

[245] K. R. Karkera. *Building probabilistic graphical models with Python.* Packt Publishing Ltd, 2014.

[246] G. Van den Broeck, K. Mohan, A. Choi, A. Darwiche, and J. Pearl. Efficient algorithms for bayesian network parameter learning from incomplete data. Meila, Marina, 2015.

[247] C. M. Bishop, D. Spiegelhalter, and J. Winn. Vibes: A variational inference engine for bayesian networks. In *Advances in neural information processing systems*, pages 793–800, 2003.

[248] C. R. Shelton, Y. Fan, W. Lam, J. Lee, and J. Xu. Continuous time bayesian network reasoning and learning engine. *Journal of Machine Learning Research*, 11(Mar):1137–1140, 2010.

[249] S. E. Shimony. A probabilistic framework for explanation. 1992.

[250] E. Charniak and S. E. Shimony. Cost-based abduction and map explanation. *Artificial Intelligence*, 66(2):345–374, 1994.

[251] B. K. Sy. Reasoning mpe to multiply connected belief networks using message passing. In *AAAI*, pages 570–576, 1992.

[252] H. L. Bodlaender, F. van den Eijkhof, and L. C. van der Gaag. On the complexity of the mpa problem in probabilistic networks. In *ECAI*, pages 675–679, 2002.

[253] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Elsevier, 2014.

[254] M. Henrion and M. J. Druzdzel. Qualtitative propagation and scenario-based scheme for exploiting probabilistic reasoning. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pages 17–32. Elsevier Science Inc., 1990.

[255] R. E. Neapolitan. *Probabilistic reasoning in expert systems: theory and algorithms.* CreateSpace Independent Publishing Platform, 2012.

[256] C. Lacave and F. J. Díez. A review of explanation methods for bayesian networks. *The Knowledge Engineering Review*, 17(2):107–127, 2002.

[257] E. Santos Jr. On the generation of alternative explanations with implications for belief revision. In *Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence*, pages 339–347. Morgan Kaufmann Publishers Inc., 1991.

[258] J. Kwisthout. Most frugal explanations in bayesian networks. *Artificial Intelligence*, 218:56–73, 2015.

[259] J. E. Hopcroft, R. Motwani, and J. D. Ullman. Automata theory, languages, and computation. *International Edition*, 24:19, 2006.

[260] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[261] J. Y. Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

A Appendix A: Case Study Full Evaluation

In the following all results of the case study are shown. The discussion of this data set was already introduced on a subset of the following material in Chapter 9. The full results are presented here

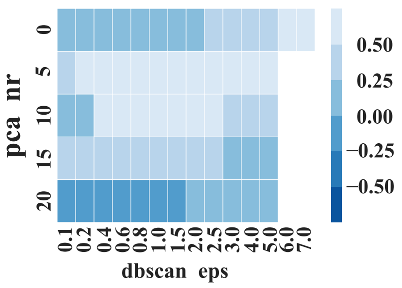
A.1 ACC

A.1.1 TV Clustering

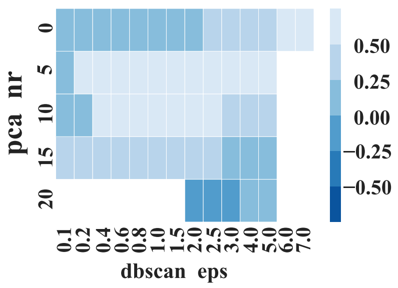
At first, TV clustering is performed with various parameters.

A.1.1.1 Hyperparameter Estimation

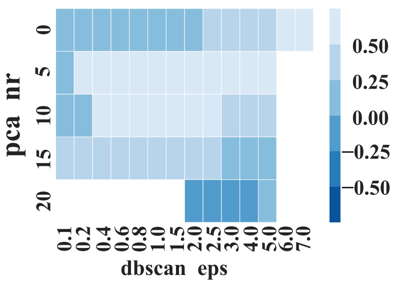
Silhouette Index, for various numbers of PCA components and various ϵ: Each plot has a fixed number set for $MinSamples = m$, when using DBSCAN



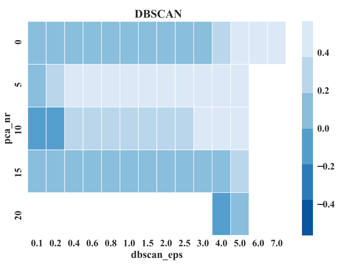
$m = 2$



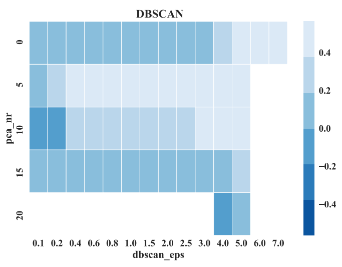
$m = 3$



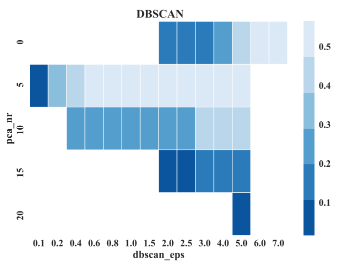
$m = 4$



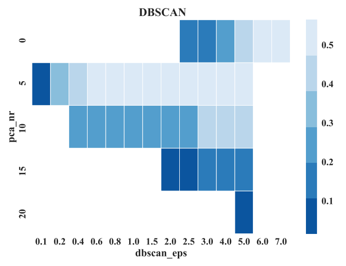
$m = 5$



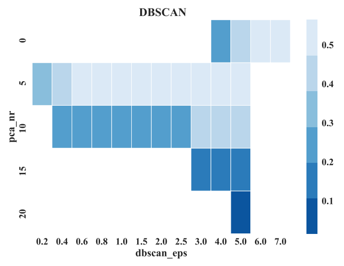
$m = 6$



$m = 7$

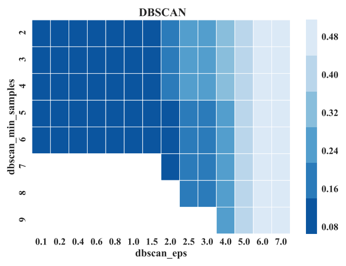


$m = 8$

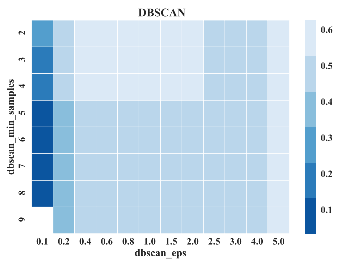


$m = 9$

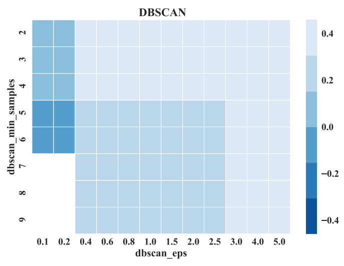
Silhouette Index, for various $MinSamples$ and various ϵ: Each plot has a fixed number set for n_{PCA}, when using DBSCAN



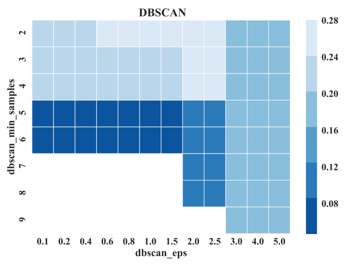
No PCA



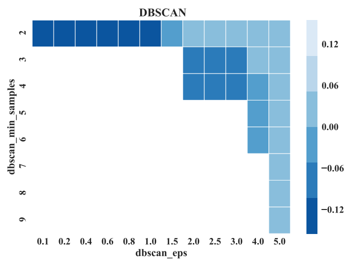
$n_{PCA} = 5$



$n_{PCA} = 10$

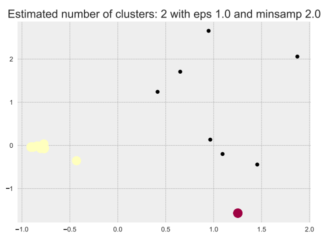


$n_{PCA} = 15$

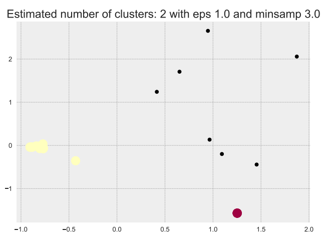


$n_{PCA} = 20$

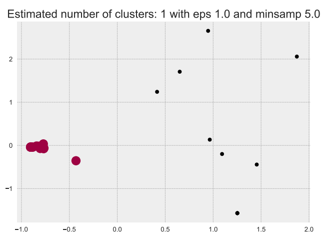
Effect of increasing $MinSamples$: Each plot shows the two most important PCA components. Further it has the optimal values for number of PCA and ϵ fixed, which is $n_{PCA} = 5$ and $\epsilon = 1.0$.



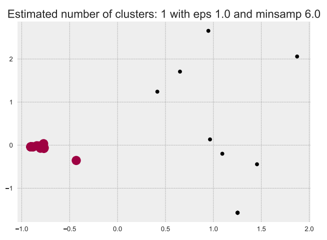
MinSamples = 2



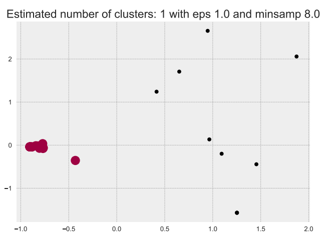
MinSamples = 3



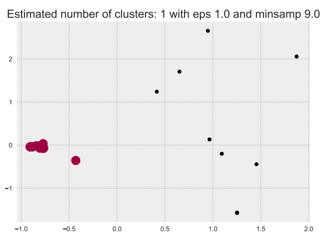
MinSamples = 5



MinSamples = 6

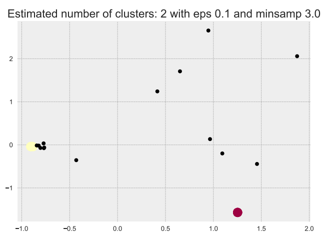


MinSamples = 8

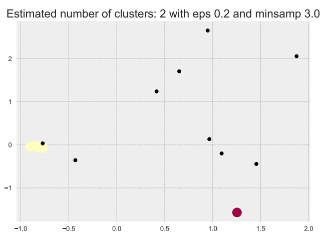


MinSamples = 9

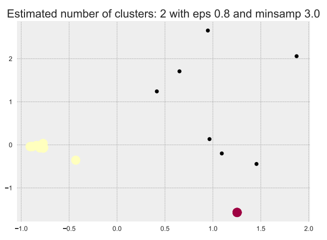
Effect of increasing ϵ: Each plot shows the two most important PCA components. Further it has the values for $MinSamples$ and ϵ fixed, which is $MinSamples = 3$ and $n_{PCA} = 5$.



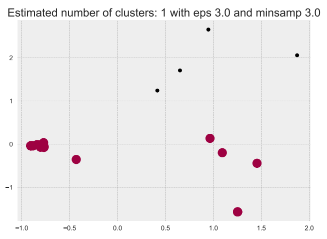
$\epsilon = 0.1$



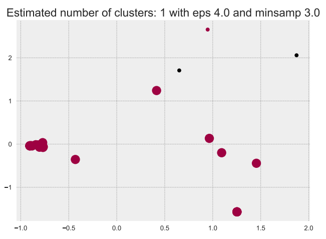
$\epsilon = 0.2$



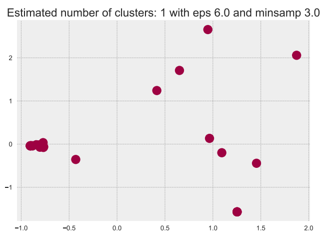
$\epsilon = 0.8$



$\epsilon = 3.0$



$\epsilon = 4.0$



$\epsilon = 6.0$

A.1.1.2 Found clusters

The algorithm found 3 clusters, including one cluster that we refer to as fragmentary which contains all TVS that were considered noise during clustering. Those clusters could be named as (1) display information, (2) car state information and a fragmentary cluster with (3) movement information.

A.1.2 Reduced Clusters

The found clustering are given as an example in this case. The selected reduced TVs used for further inspection are a subset of those TVs that were chosen based on this clustering and were carefully selected by an expert. Thus, here there is only one TV

cluster called *TV Cluster 1*.
Statistics of chosen clusters:
TV Cluster 1:

- **Number of TVs [num / nom / bin / ord]:** 18 [0/2/16/0]
- **Number of samples [num / nom / bin / ord]:** 16663 [699/15964/0/0]
- **Mean gap:** 0.306 s
- **Std. gap:** 2.748 s

A.1.3 Segmentation Clustering:

A.1.3.1 Hyperparameter Estimation

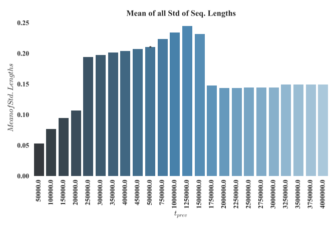
For this data set the targeted approach was used once for the activation and for the deactivation. Thus, we assume that two clusters are found. Those are Cluster 1 as ACC deactivation and cluster 2 as ACC activation. Per data set hyperparameters are found as described in the next section.

A.1.4 Specification Extraction

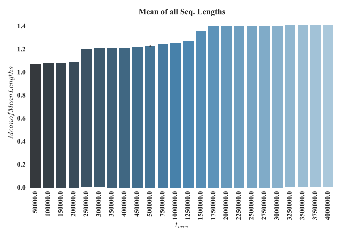
On a subset of the segments found during Segmentation Clustering Specification Mining with BaySpec and MPE is used to find specifications.

A.1.5 TV Cluster 1 - Segment Group 0

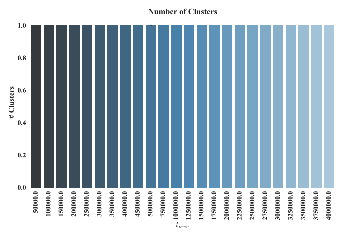
A.1.5.1 Segmentation Clustering



Mean Std. of Seq. Lengths



Mean Seq. Lengths



Number of Clusters

Best parameters maximize the length of found sequences while having a small Std. deviation of sequence lengths, which indicates consistency. This is given at the drop point at around $t_{prev} = 2$ seconds. The number of segment clusters is one here as we are considering all segments before the target change as one cluster. Those parameters result in clusters with the following statistics.

- **Number of MSSs:** 3808
- **Mean of Means of Timespans per TV:** 0.015 s
- **Mean of Std. of Timespans per TV:** 0.0005 s

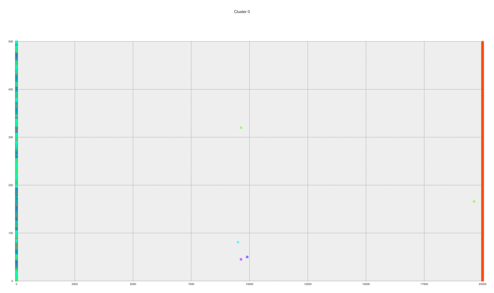


Figure A.29: Example of segment groups of MSSs.

- **Number of TVs:** 11
- **Mean Sequence Lengths per TV:** 1.09
- **Mean of Std. of Sequence Lengths per TV:** 0.01

Examples of resulting segment groups are given in Figure A.29, where each color indicates a TV with a certain value at a given time and each dimension along the y axis corresponds to one observed MSS.

A.1.5.2 Model Training

As the episodes contain less variation in the sequences we choose parameters that lead to a structure of maximal size, but reduced by temporal logic as part of TrieDiscover. Thus, $k = 0.00$, $\alpha = 1.0$ and $\chi_{th} = 0.2$ are used, which gives the structure given in Figure A.30.
For parameter estimation EM is used with 1000 MCMC samples and 5 iterations.

A.1.5.3 Specification Extraction

With this BaySpec is used to produce specifications as shown in Figure A.30. 24 specifications with a threshold $p_{min} > 0.6$ were found. The found specifications cannot be exactly published here as those are proprietary.
MPE is used to extract snapshots of MSSs that represent dominant and thus, specification behavior. Sorted by likelihood the following specifications were found (the exact names of TVs were altered):

- Deactivation, if the driver was braking given a certain speed:
  G(((speed:b or brake-intensity:d) $\rightarrow$ X(F(ACC:deactivated))))
  This specification occurred multiple times with varying brake levels of a, b, c and d and for various speeds. The same specification occurred for various other TVs where in the first part, i.e. (speed:b or brake-intensity:d), speed got replaced by other initial TVs, e.g. (button1:not pressed or brake-intensity:d). This seemed to be the dominating reason for deactivation.

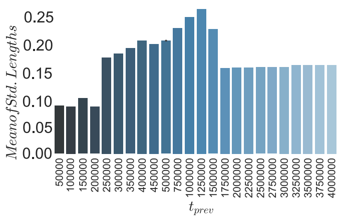
The two dominating MPEs are given in the following where dt is specified in micro seconds and a subset of TVs is named.
Example 1: The first most likely found structures vary in speed (a,b,c,d), while deactivation occurs because of braking. This makes sense as deactivation occurs at any speed.
ACC - dt=0.0 - activated - dt=20000 - deactivated
preceding - dt=0.01 - close
TV-1 - dt=0.1 - state A
TV-2 - dt=0.10 - state A
TV-3 - dt=0.09 - state A
setButton - dt=0.04 - not pressed
TV-4 - dt=0.26 - state A
speed - dt=0.05 - a
brake-intensity - dt=0.0 - d
brake-state - dt=0.0 - 1
warnings - dt=0.0 – none

Example 2: Manual deactivation.
ACC - dt=0.0 - activated - dt=20001 - deactivated
preceding - dt=0.0 - close
TV-1 - dt=0.09 - state A
TV-2 - dt=0.0 - state A
TV-3 - dt=0.0 - state A
setButton - dt=0.0 - pressed
TV-4 - dt=0.0 - state A
speed - dt=0.0 - b
brake-intensity - dt=0.29 - d
brake-state - dt=0.0 - 1
warnings - dt=0.14 – none

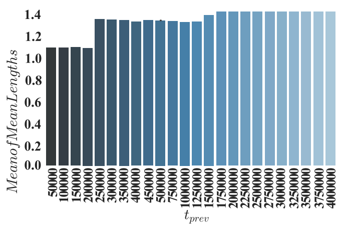
A.1.6 TV Cluster 1 - Segment Group 1

Here the same argumentation holds as in the previous subsection.

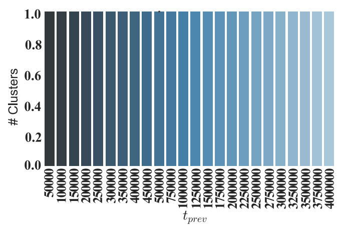
A.1.6.1 Segmentation Clustering



Mean Std. of Seq. Lengths



Mean Seq. Lengths



Number of Clusters

Best parameters are $t_{prev} = 2$, which result in clusters with the following statistics.

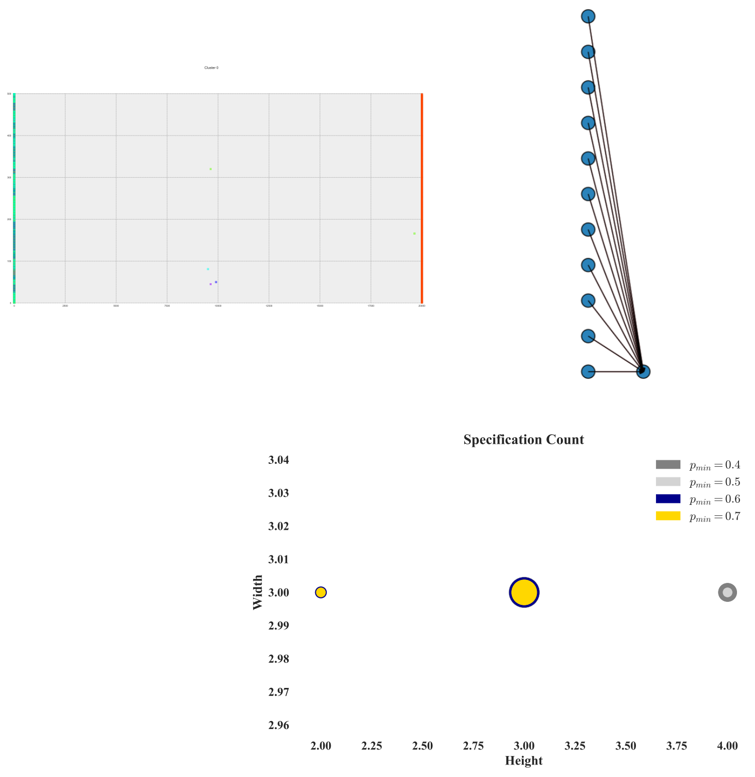


Figure A.30: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

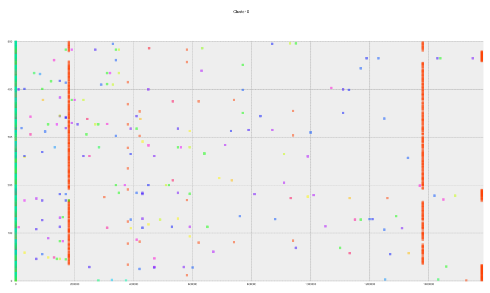


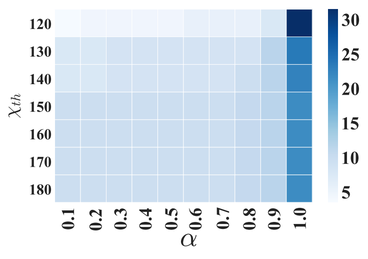
Figure A.34: Example of segment groups of MSSs.

- **Number of MSSs:** 3808
- **Mean of Means of Timespans per TV:** 0.504 s
- **Mean of Std. of Timespans per TV:** 0.313 s
- **Number of TVs:** 11
- **Mean Sequence Lengths per TV:** 1.12
- **Mean of Std. of Sequence Lengths per TV:** 0.14

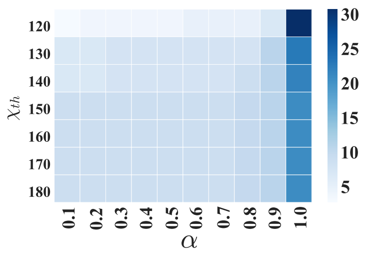
Examples of resulting segment groups are given in Figure A.34.

A.1.6.2 Model Training

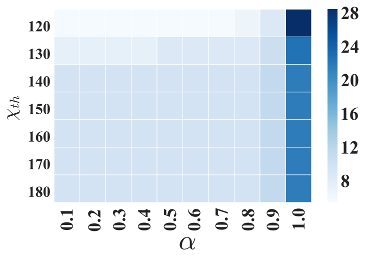
Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.



$k = 0.00$



$k = 0.05$



$k = 0.15$

$k = 0.00$, $\alpha = 1.0$ and $\chi_{th} = 0.2$ are used, which gives the structure given in Figure A.38.

A.1.6.3 Specification Extraction

With this BaySpec is used to produce specifications as shown in Figure A.38. 224 specifications with a threshold $p_{min} > 0.6$ were found. The found specifications cannot be exactly published here as those are proprietary.

MPE is used to extract snapshots of MSSs that represent dominant and thus, specification behavior. Sorted by likelihood the following specifications were found (the exact names of TVs were altered):

- Some meaningless specifications were found, which resemble system states that were present during activation. While those specifications are not wrong they can be considered irrelevant. Examples are the state of fact that the display shows ones a warning ones no warning before a no preceding car detection is shown and ACC activates. Thus, the warning does not have an influence and is not the cause of an activation.
  G((warnings:none → X(F((preceding:none and X(F(ACC:activated)))))))
  G((warnings:showing → X(F((preceding:none and X(F(ACC:activated)))))))
- Further, a main pattern was the preceding car information that occurred in most specifications, which is triggered in two cases. First, if the brake was released
  G((brake:deactivated → X(F((preceding:none and X(F(ACC:activated)))))))
- Second, if a button was pressed, which can be different combinations, including the resume button
  G((resumeButton:pressed → X(F((preceding:none and X(F(ACC:activated)))))))
  and the Set button
  G((setButton:pressed → X(F((preceding:none and X(F(ACC:activated)))))))
- Also, various combinations before activation are checked. Those include multiple braking states that are allowed, which are checked similar to the first case.
  G((brake-state:1 → X(F((preceding:none and X(F(ACC:activated)))))))
  G((brake-state:3 → X(F((preceding:none and X(F(ACC:activated)))))))
  In those cases an activation is only allowed if any of those states are given. This is not directly checkable with this formula, as it is a reverse action that needs to be checked once an activation occurred instead of the inverted case. Thus, for this case either an adaption of specifications needs to be performed or prior filtering for activations needs to be performed and checking needs to be performed on activation MSSs only.

In higher likelihood specifications there are little false positives. However, often redundancy is present as was given in the case of above warning case.

The dominating MPEs are given in the following where dt is specified in micro seconds and a subset of TVs is named.
Example 1: The first most likely found structures captures an deactivation by braking followed by an activation.
ACC - dt=0.0 - activated - dt=0.0 - deactivated - dt=1373978 - activated
preceding - dt=0.0 - close - dt=20001 - none
TV-1 - dt=0.0 - state A
TV-2 - dt=0.0 - state A
TV-3 - dt=0.0 - state A
setButton - dt=0.3 - not pressed
TV-4 - dt=0.0 - state A

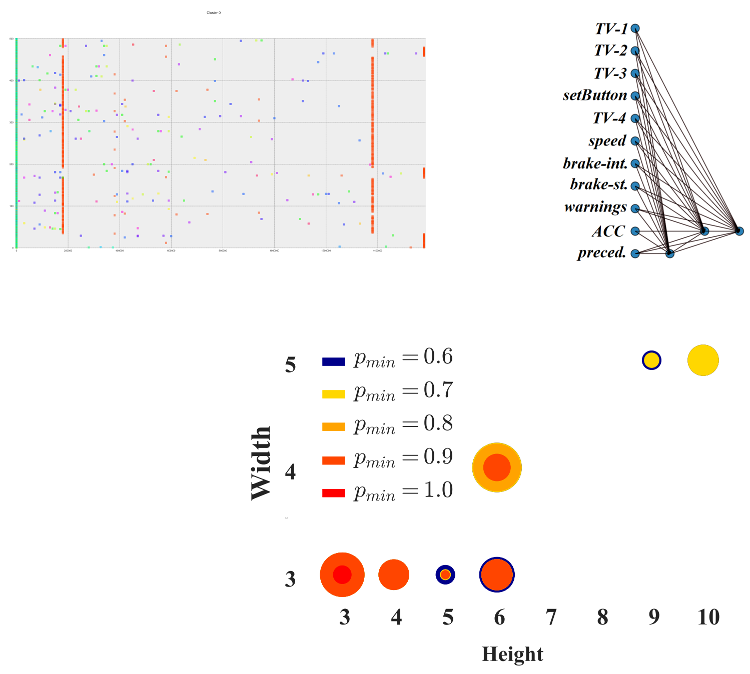


Figure A.38: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

speed - dt=0.2 - a
brake-intensity - dt=0.4 - d
brake-state - dt=0.0 - 1
warnings - dt=0.0 - none

Example 2: Here an activation is shown, that results from pressing the Set button.
ACC - dt=0.1 - activated - dt=0.0 - activated - dt=0.0 - activated
preceding - dt=0.0 - close - dt=0.0 - none
TV-1 - dt=0.1 - state A
TV-2 - dt=0.0 - state A
TV-3 - dt=0.0 - state A
setButton - dt=0.4 - pressed
TV-4 - dt=0.0 - state A
speed - dt=0.0 - b
brake-intensity - dt=0.0 - d
brake-state - dt=0.0 - not pressed
warnings - dt=0.0 - none

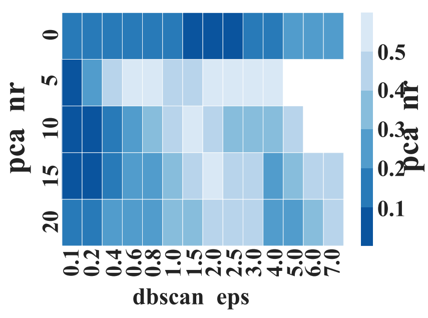
A.2 Lights

A.2.1 TV Clustering

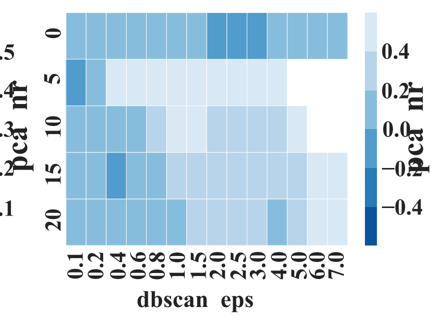
At first, TV clustering is performed with various parameters.

A.2.1.1 Hyperparameter Estimation

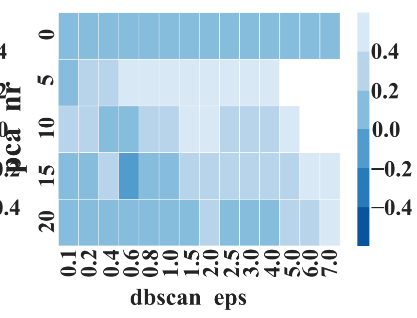
Silhouette Index, for various numbers of PCA components and various ϵ: Each plot has a fixed number set for $MinSamples = m$, when using DBSCAN



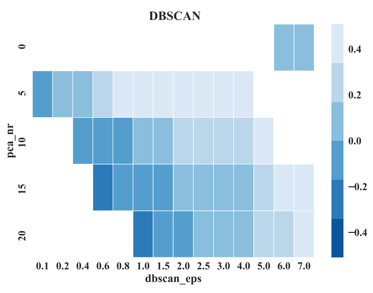
$m = 2$



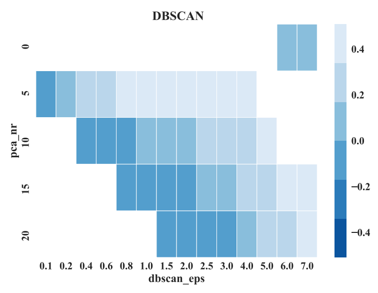
$m = 3$



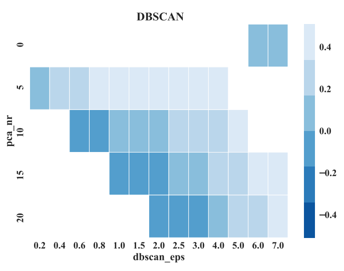
$m = 4$



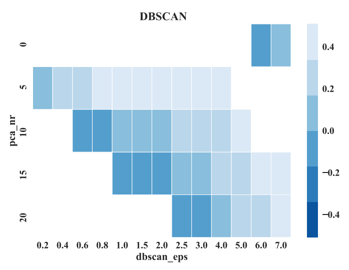
$m = 5$



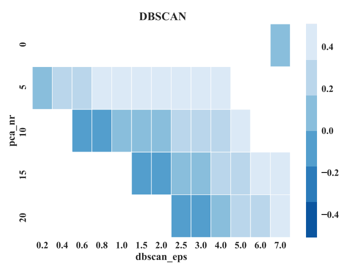
$m = 6$



$m = 7$

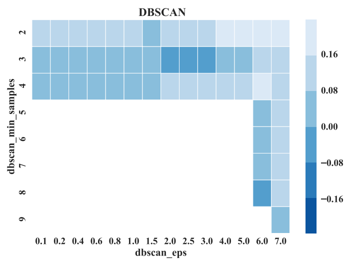


$m = 8$

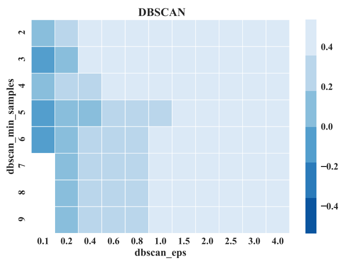


$m = 9$

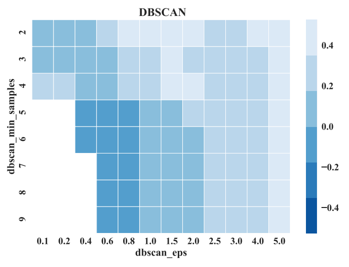
Silhouette Index, for various $MinSamples$ **and various ϵ:** Each plot has a fixed number set for n_{PCA}, when using DBSCAN



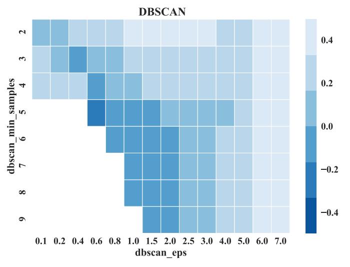
No PCA



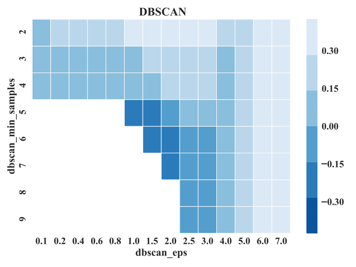
$n_{PCA} = 5$



$n_{PCA} = 10$

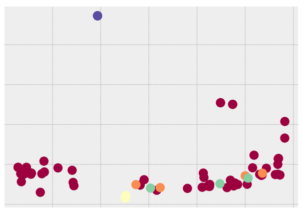


$n_{PCA} = 15$

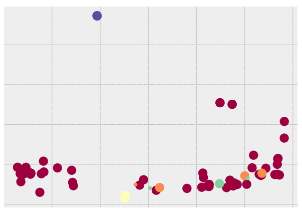


$n_{PCA} = 20$

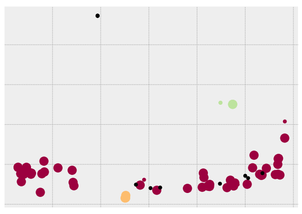
Effect of increasing $MinSamples$**:** Each plot shows the two most important PCA components. Further it has the optimal values for number of PCA and ϵ fixed, which is $n_{PCA} = 5$ and $\epsilon = 1.5$.



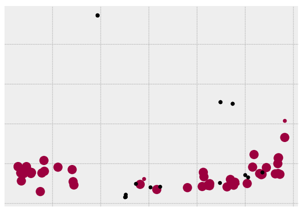
MinSamples = 2



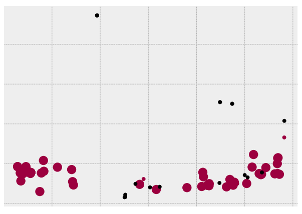
MinSamples = 3



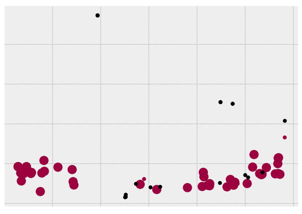
MinSamples = 5



MinSamples = 6

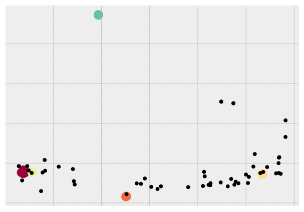


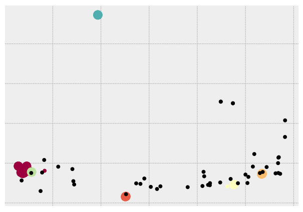
MinSamples = 8

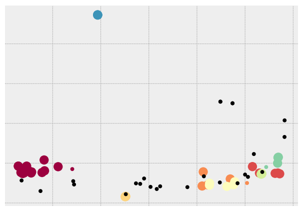


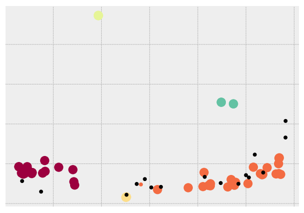
MinSamples = 9

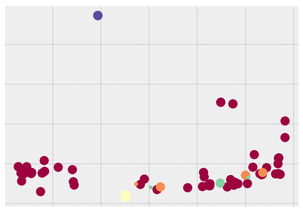
Effect of increasing ϵ**:** Each plot shows the two most important PCA components. Further it has the values for $MinSamples$ and ϵ fixed, which is $MinSamples = 3$ and $n_{PCA} = 5$.

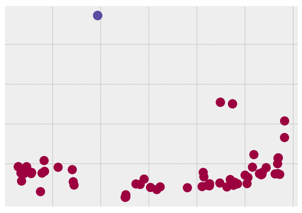

$\epsilon = 0.1$


$\epsilon = 0.2$


$\epsilon = 0.4$


$\epsilon = 0.8$


$\epsilon = 1.5$


$\epsilon = 3.0$

A.2.1.2 Found clusters

The algorithm found 5 clusters, including one cluster that we refer to as fragmentary which contains all TVS that were considered noise during clustering. Those clusters could be named as (1) light behavior including states of the indicator or front lights, (2) detection of traffic (3) engine movement (4) environment values and a fragmentary cluster with (5) steady behavior.

A.2.1.3 Reduced Clusters

Based on the found clustering a subset of TVs was selected by experts for further investigation.
Statistics of chosen clusters:
TV Cluster 1:

- **Number of TVs [num / nom / bin / ord]:** 14 [0/10/4/0]
- **Number of samples [num / nom / bin / ord]:** 54839 [0/43533/11306/0]
- **Mean gap:** 16.102 s
- **Std. gap:** 23.940 s

TV Cluster 2:

- **Number of TVs [num / nom / bin / ord]:** 32 [5/17/9/1]
- **Number of samples [num / nom / bin / ord]:** 168339 [37326/115308/3281/12424]

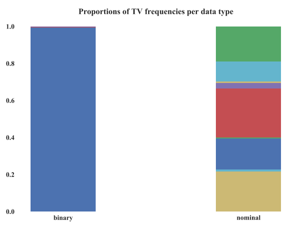


Figure A.64: TV Cluster 1: Proportions of TVs

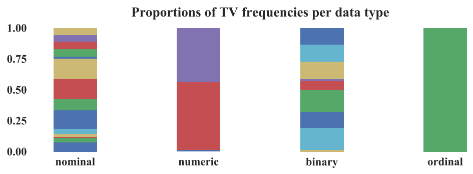


Figure A.65: TV Cluster 2: Proportions of TVs

Figure A.66: Proportion of TVs per Cluster

- **Mean gap:** 10.122 s
- **Std. gap:** 19.527 s

The proportion of occurrence frequencies per TV and data type is shown in Figure A.66.

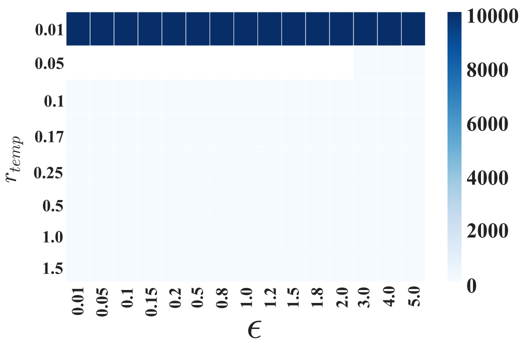
A.2.2 Segmentation Clustering

For this data set the range segmentation clustering is used. Per data set and cluster of TVs hyperparameters are found as described in this section, with statistics shown in the following.

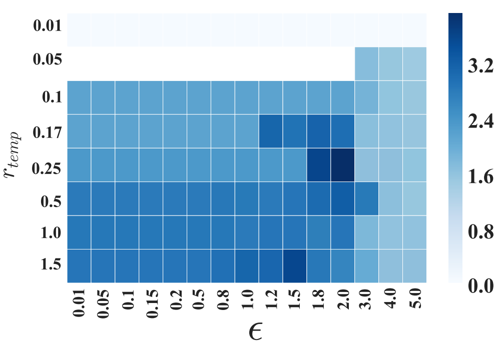
A.2.2.1 Hyperparameters

When varying the parameters of range segmentation, the mean of all sequence lengths per TV, the mean of standard deviations of all sequence lengths per TV, the number of found clusters and the silhouette index is measured. The results of this evaluation is shown in the following, for each TV cluster.

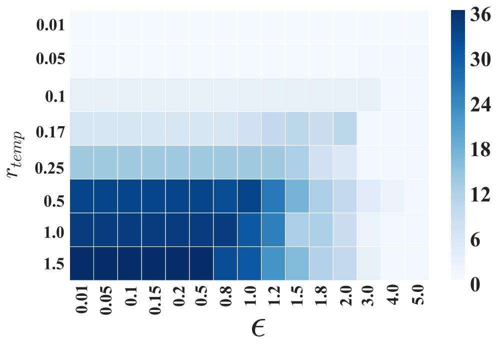
TV Cluster 1:



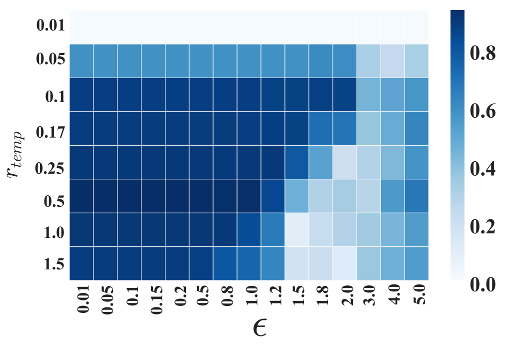
Mean of Std. of Sequence lengths per TV



Mean of Sequence lengths per TV

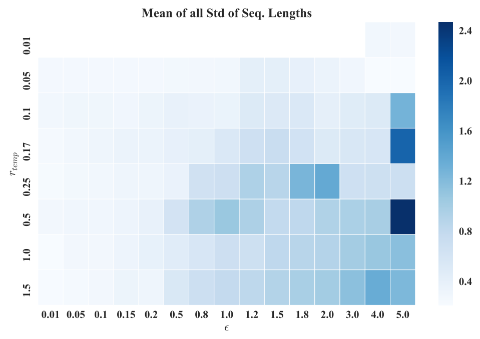


Number of Clusters found

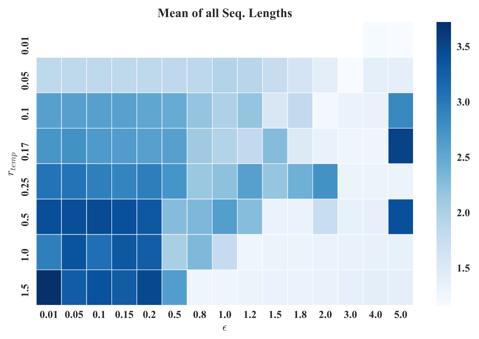


Silhouette index

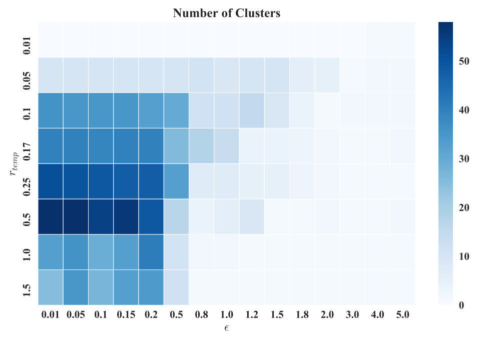
TV Cluster 2:



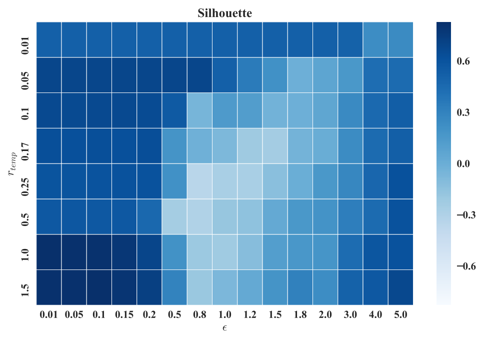
Mean of Std. of Sequence lengths per TV



Mean of Sequence lengths per TV



Number of Clusters found



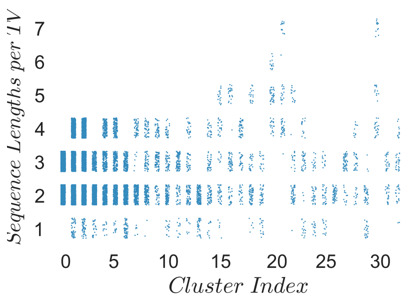
Silhouette index

The goal is to find parameters producing many clusters and are as long as possible, while keeping the standard deviation low between MSSs. Thus, for TV cluster 1 $\epsilon = 0.5$ and $r_{temp} = 0.5$ and for TV cluster 2 $\epsilon = 1.0$ and $r_{temp} = 0.01$ were chosen.

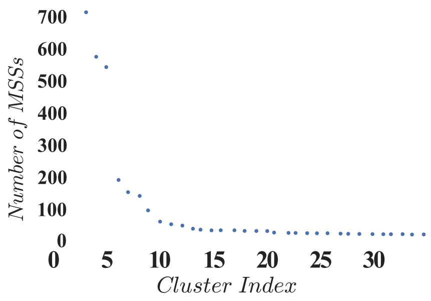
A.2.2.2 Statistics

For each sequence the length of a sequence of a TV was measured per cluster and resembles one data point in the first statistics plot. For better readability a jitter was added. E.g. In MSSs of cluster 0 there are only TVs that have sequence lengths of length 2 and 3. The second plot shows the number of MSSs per cluster. The third plot shows the number of TVs per MSS in the group of segments as one data point, again with jitter added. Lastly, examples of found MSSs are shown, where the color of a data point indicates the occurrence of a TV with a certain outcome.

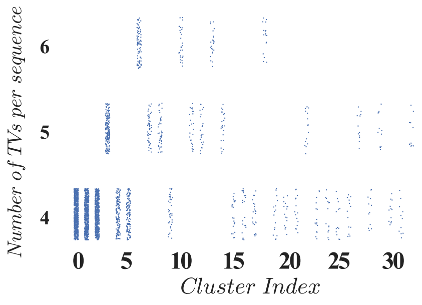
TV Cluster 1:



Sequence lengths per TV

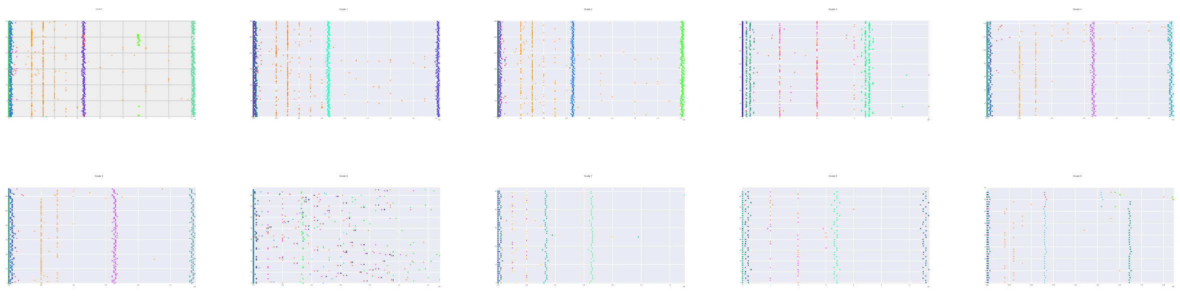


Number of MSSs

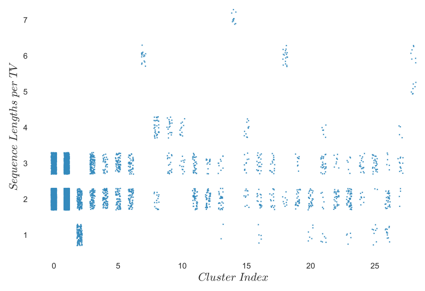


Number of TVs per sequence

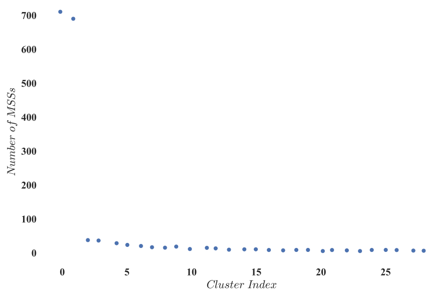
First 10 clusters:



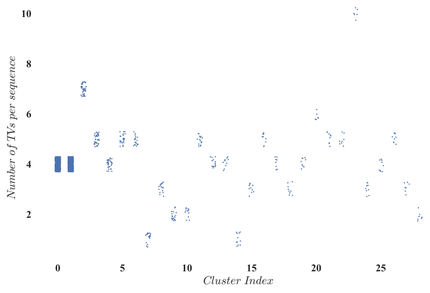
TV Cluster 2:



Sequence lengths per TV



Number of MSSs



Number of TVs per sequence

First 10 clusters:

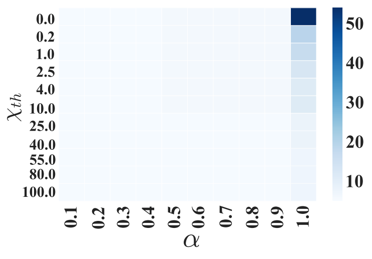
A.2.3 Specification Extraction

Next, on a subset of the segments found during Segmentation Clustering Specification Mining with BaySpec and MPE is used to find specifications. Here, only a subset of clusters and segments is considered.

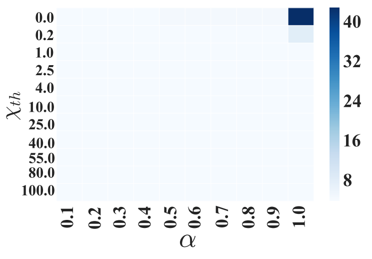
A.2.4 TV Cluster 1 - Segment Group 0

A.2.4.1 Model Training

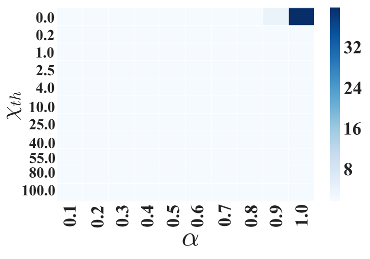
Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha = 1.0$ and $\chi_{th} = 0.2$, which gives the structure given in Figure A.84.

A.2.4.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.84. 104 specifications with a threshold $p_{min} > 0.6$ were found.
This segment resembles the short activation of the indicators for three seconds. Sorted by likelihood the following specifications were found (the exact names of TVs were altered). The three dominant observations are the activations of left and right indicator as well as the synchronization information sent, followed by a deactivation (which of course has to occur after 3 seconds).

- The right activation is
  G(((bar:tip up or state:both off) → X(G((state:right on → X(state:both off))))))
- The left activation is
  G(((bar:tip down or state:both off) → X(G((state:left on → X(state:both off))))))
- The synchronization is
  G((sync:indicator continues → X(G((sync:start cycle → X(F((sync:initialize or sync:indicator continues))))))))
- Further specifications include the handle bar variation, from pressed to not pressed or the variant of a indicator changing to the defect state.

The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named.
Example 1: The first most likely found structures captures the activation of the right indicator, which turns on and off again. The same is found for the left indicator.
bar - dt=0.0 - tip up - dt=0.0 - tip up - dt=716707958 - not pressed
sync - dt=0.0 - indicator continues - dt=22022086 - start cycle - dt=0.0 - initialize
state - dt=0.0 - both off - dt=15745713 - right on - dt=1740429348 - right off
sync2 - dt=0.0 - not indicating - dt=17029405 - indicating - dt=1586328954 - not indicating

Example 2: Here, the defect case is shown.
bar - dt=0.0 - tip up - dt=0.0 - tip down - dt=555459463 - not pressed

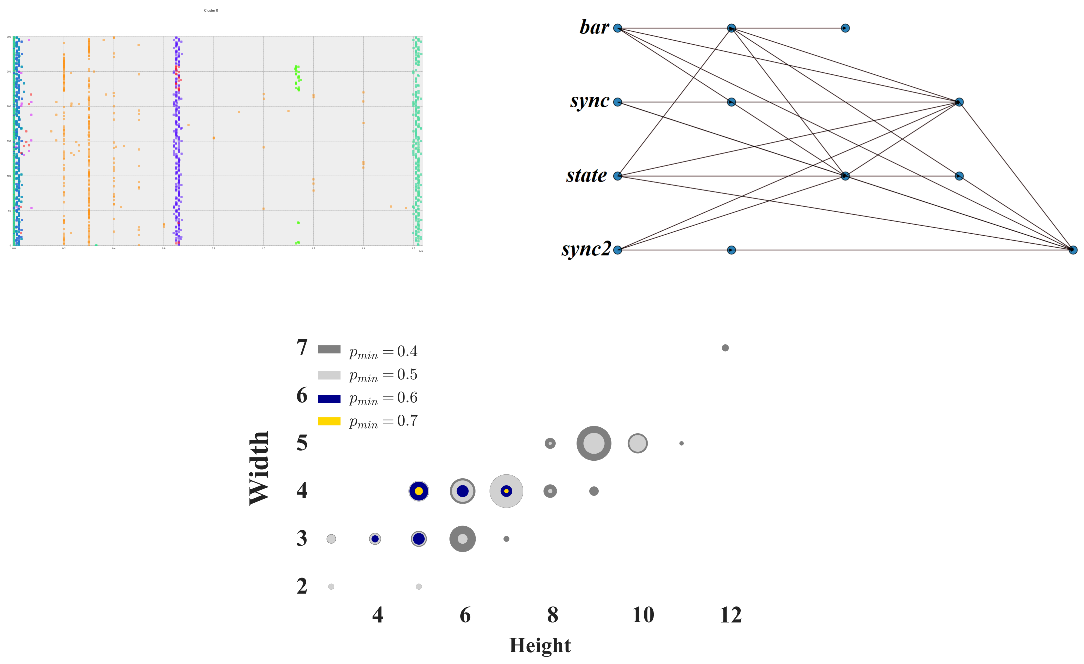


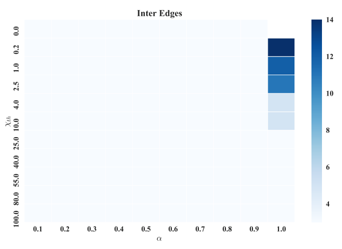
Figure A.84: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

sync - dt=0.0 - indicator continues - dt=11216975 - start cycle - dt=479977508 - indicator continues
state - dt=0.0 - both off - dt=16188873 - left on - dt=1749651319 - both off
sync2 - dt=0.0 - not indicating - dt=3574150 - indicating - dt=640113760 - indication defect

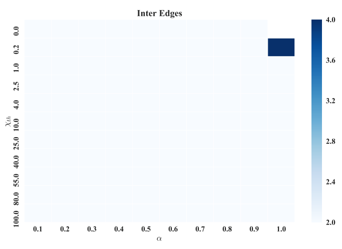
A.2.5 TV Cluster 1 - Segment Group 3

A.2.5.1 Model Training

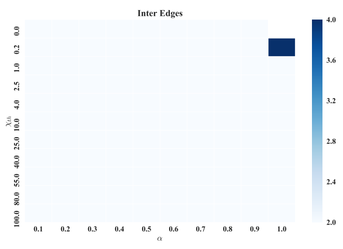
Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha = 1.0$ and $\chi_{th} = 0.2$, which gives the structure given in Figure A.88.

A.2.5.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.88. 84 specifications with a threshold $p_{min} > 0.6$ were found.
This segment resembles the permanent activation of the indicators. Sorted by likelihood the following specifications were found (the exact names of TVs were altered). The dominant specifications include on the one hand the over tipping of the handle bar which starts another synchronization and activates the indicator, and on the other hand the allowed states of the vehicle during activation.

- The vehicle state case include
  G(((state:both off or vehicle:parking-idle)) → X(vehicle:indicating)))
  G(((vehicle:parking-idle or sync:indicator continues) → X(G((sync:new cycle → X(sync:indicator continues))))))
- The activation synchronization
  G(((type:not permanent or sync:new cycle) → X(F(type:permanent))))
- The upper over pressing of the handle bar
  G(((sync:new cycle or bar:tip up) → X(F((state:right on and X(F(bar:over press up)))))))
- The lower over pressing of the handle bar
  G(((sync:new cycle or bar:tip down) → X(F((state:left on and X(F(bar:over press down)))))))
- Further specifications include variants of those activations, such as combinations of synchronizations, bar activity and indicator state.

The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named.
Example 1: The long indication of the left indicator is shown. The right indicator was discovered as second most likely.
bar - dt=0.3 - tip down - dt=0.0 - tip down - dt=110532311 - overpress down
sync - dt=0.0 - indicator continues - dt=13123695 - start cycle - dt=329742908 - indicator continues
type - dt=0.2 - not permanent - dt=332681727 - permanent
state - dt=0.1 - both off - dt=15977380 - left on
sync2 - dt=0.0 - not indicating - dt=16465218 - indicating
Other likely MPEs are variations of this.

A.2.6 TV Cluster 2 - Segment Group 3

A.2.6.1 Model Training

Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.

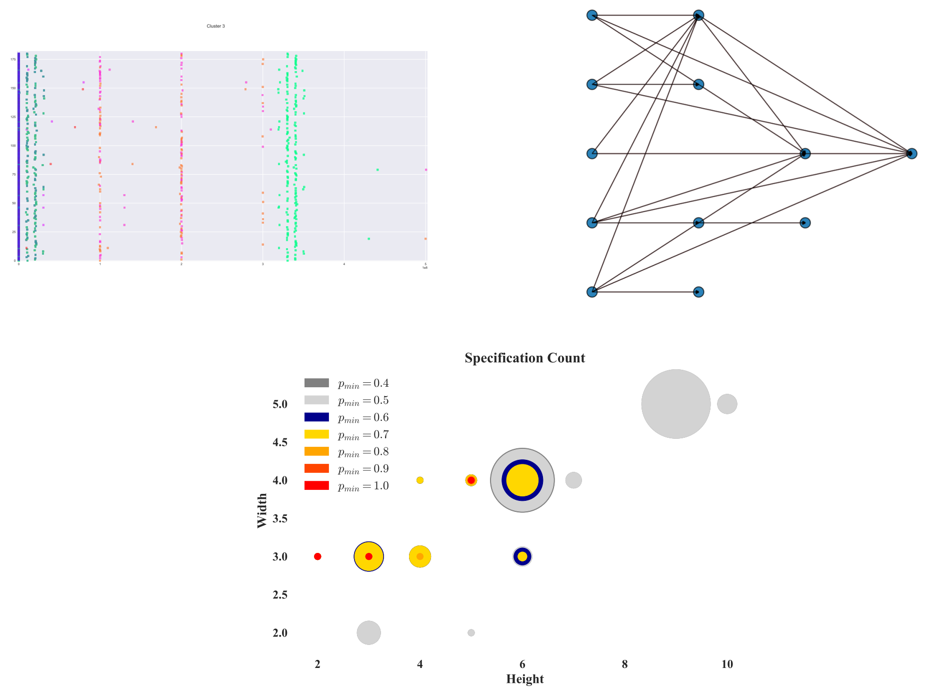
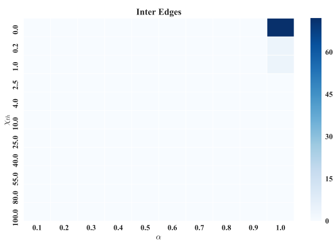
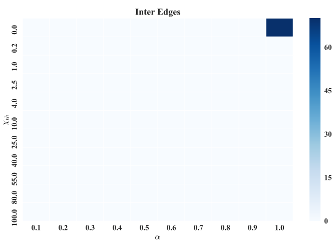


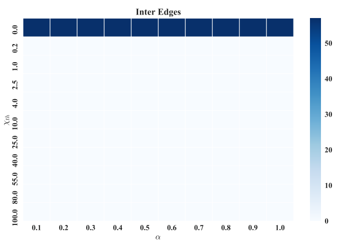
Figure A.88: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha = 1.0$ and $\chi_{th} = 0.2$, which gives the structure given in Figure A.92.

A.2.6.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.92. 12 specifications with a threshold $p_{min} > 0.6$ were found.
This segment resembles the movement of the lights on approaching traffic. Sorted by likelihood the following specifications were found (the exact names of TVs were altered). The dominant specifications are the horizontal and vertical movement and combinations of vehicle states during movement.

- Light movement activates and deactivates on approaching traffic. For instance if it was already on it will remain on while it will turn on otherwise.
  G(((light-vertical-right:on or light-vertical-right:off) $\rightarrow$ X(G((light-vertical-right:off $\rightarrow$ X(light-vertical-right:on))))))
  Similar specifications are found for the left case and vertical movement.
- The allowed vehicle state when a movement of the light occurs is captured.
  G(((light-vertical:inactive or light-vertical:on) $\rightarrow$ X(F(driver:present))))
- Further specifications include the order of light activation and state information transmitted.

The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named.
Example 1: The dominant state is resembling the driver being present while driving and approaching traffic.
driver - dt=0.1 - driving - dt=869378104 - present - dt=0.1 - present
light-hor-left - dt=0.1 - inactive - dt=424301578 - active - dt=637636444 - inactive
light-hor-right - dt=0.0 - inactive - dt=77739177 - active
light-vertical-left - dt=0.0 - active - dt=0.0 - inactive - dt=378360691 - active
light-vertical-right - dt=0.0 - active - dt=1229578915 - inactive - dt=378968005 - active
Other likely MPEs are variations of this.

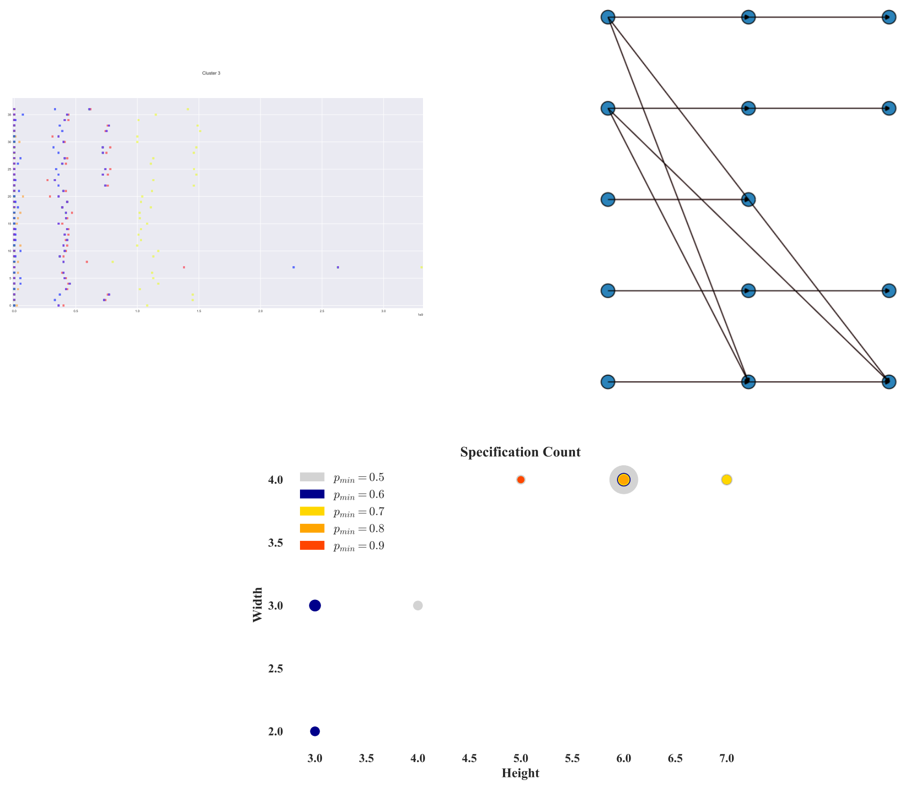


Figure A.92: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

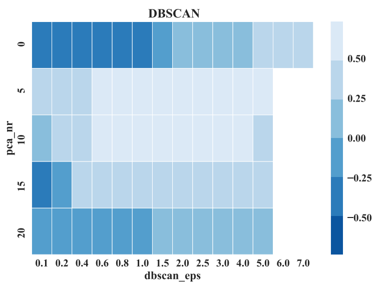
A.3 Wiper

A.3.1 TV Clustering

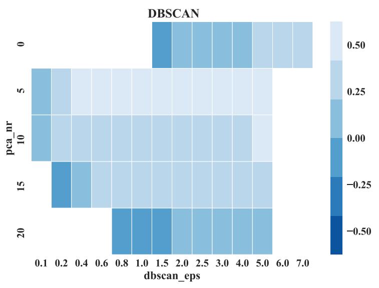
At first, TV clustering is performed with various parameters.

A.3.1.1 Hyperparameter Estimation

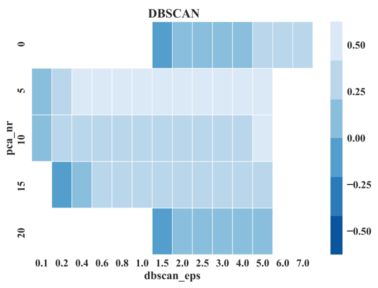
Silhouette Index, for various numbers of PCA components and various ϵ: Each plot has a fixed number set for $MinSamples = m$, when using DBSCAN



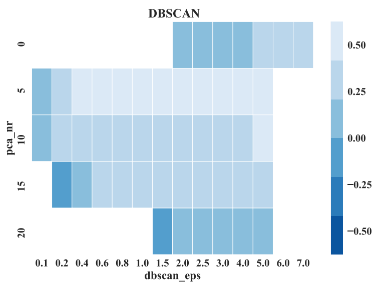
$m = 2$



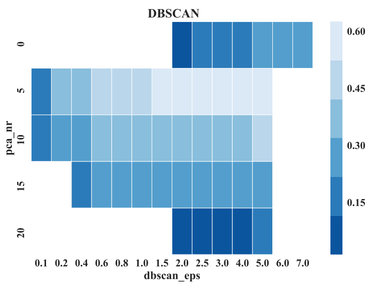
$m = 3$



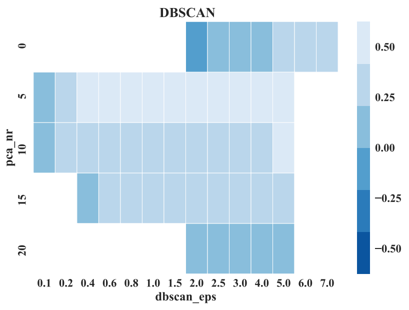
$m = 4$



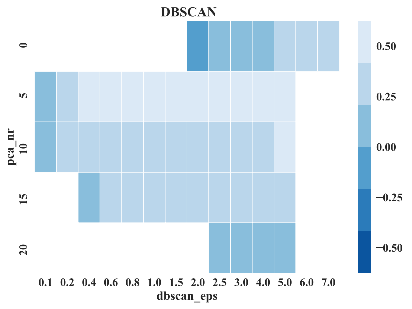
$m = 5$



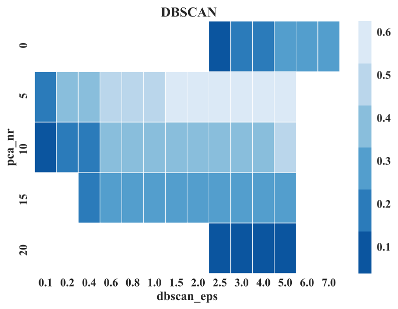
$m = 6$



$m = 7$

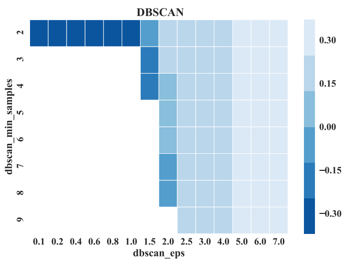


$m = 8$

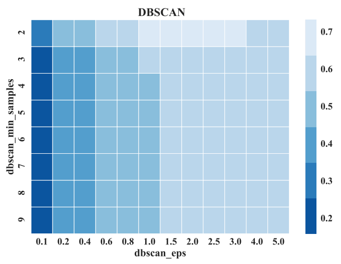


$m = 9$

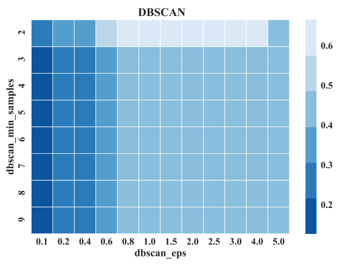
Silhouette Index, for various $MinSamples$ and various ϵ: Each plot has a fixed number set for n_{PCA}, when using DBSCAN



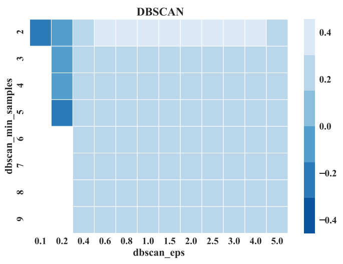
No PCA



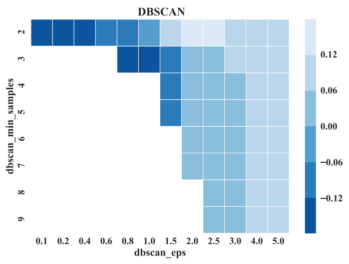
$n_{PCA} = 5$



$n_{PCA} = 10$

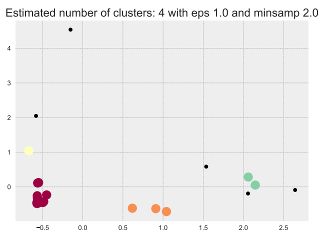


$n_{PCA} = 15$

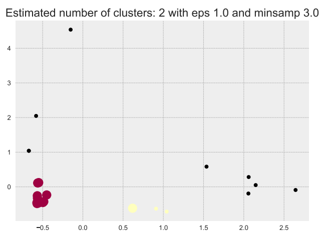


$n_{PCA} = 20$

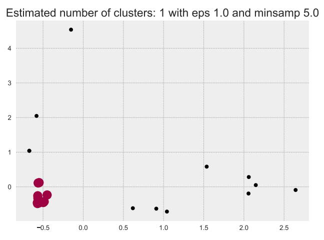
Effect of increasing $MinSamples$**:** Each plot shows the two most important PCA components. Further it has the optimal values for number of PCA and ϵ fixed, which is $n_{PCA} = 5$ and $\epsilon = 1.5$.



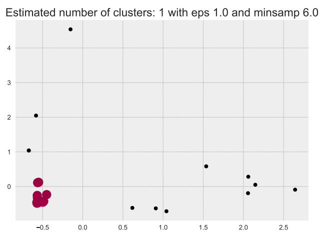
MinSamples = 2



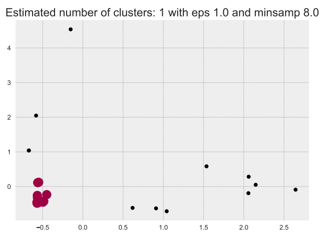
MinSamples = 3



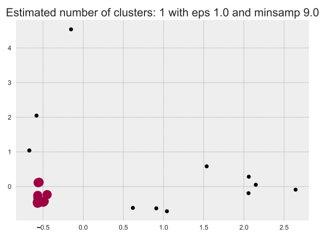
MinSamples = 5



MinSamples = 6

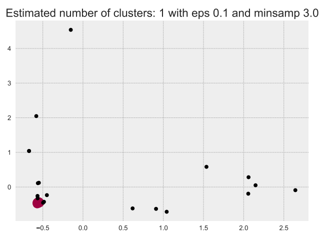


MinSamples = 8

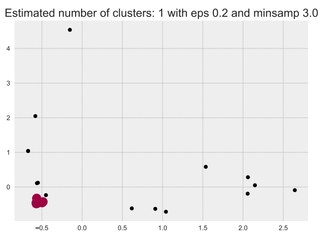


MinSamples = 9

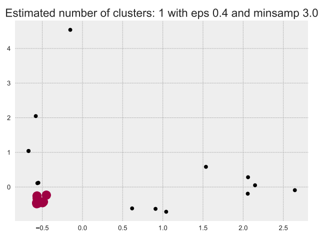
Effect of increasing ϵ**:** Each plot shows the two most important PCA components. Further it has the values for $MinSamples$ and ϵ fixed, which is $MinSamples = 3$ and $n_{PCA} = 5$.



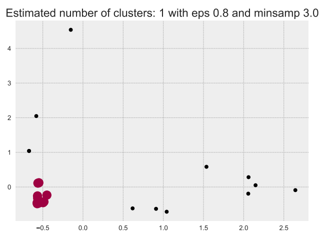
$\epsilon = 0.1$



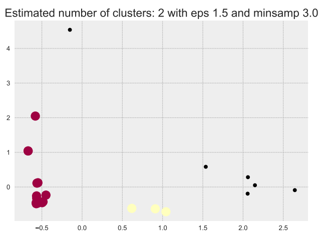
$\epsilon = 0.2$



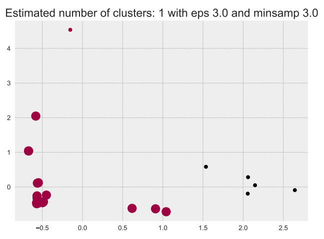
$\epsilon = 0.4$



$\epsilon = 0.8$



$\epsilon = 1.5$



$\epsilon = 3.0$

A.3.1.2 Found clusters

The algorithm found 3 clusters, including one cluster that we refer to as fragmentary which contains all TVS that were considered noise during clustering. Those clusters could be named as (1) rain sensor and wiper control, (2) wiper position and a fragmentary cluster which include (3) environmental information.

A.3.1.3 Reduced Clusters

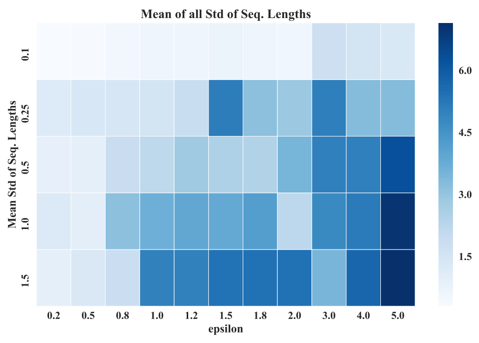
Based on the found clustering a subset of TVs was selected by experts for further investigation. In the Wiper data set this does not apply, as a preselected set of relevant TVs was considered here only. Thus, there is only one cluster here which has same statistics as the total data set.

A.3.2 Segmentation Clustering

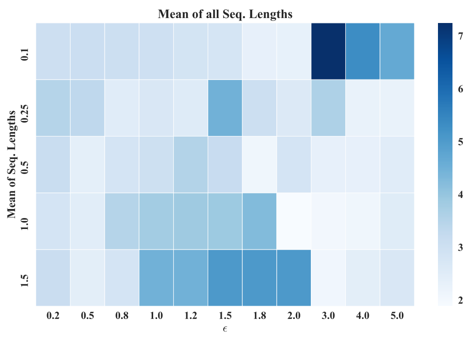
For this data set the range segmentation clustering is used. Per data set and cluster of TVs hyperparameters are found as described in this section, with statistics shown in the following.

A.3.2.1 Hyperparameters

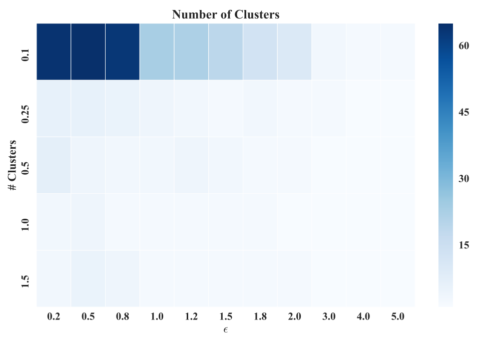
TV Cluster 1:



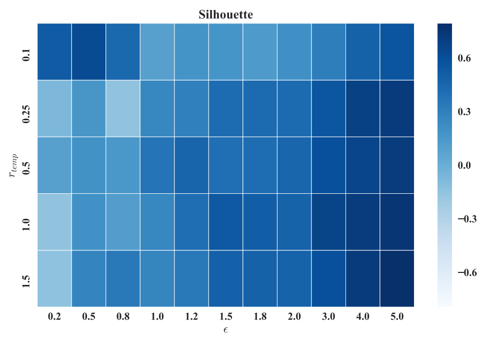
Mean of Std. of Sequence lengths per TV



Mean of Sequence lengths per TV



Number of Clusters found

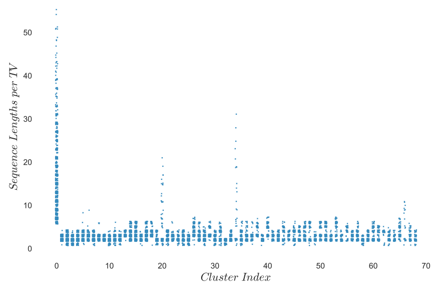


Silhouette index

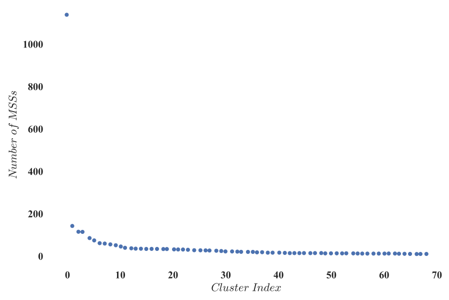
The goal is to find parameters producing many clusters and are as long as possible, while keeping the standard deviation low between MSSs. Thus, for TV cluster 1 $\epsilon = 0.8$ and $r_{temp} = 0.1$ was chosen.

A.3.2.2 Statistics

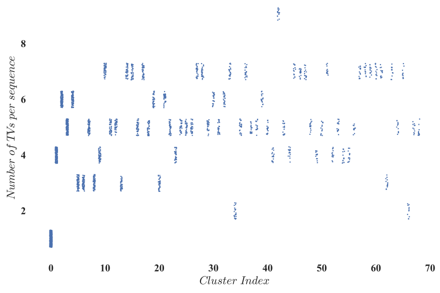
TV Cluster 1:



Sequence lengths per TV

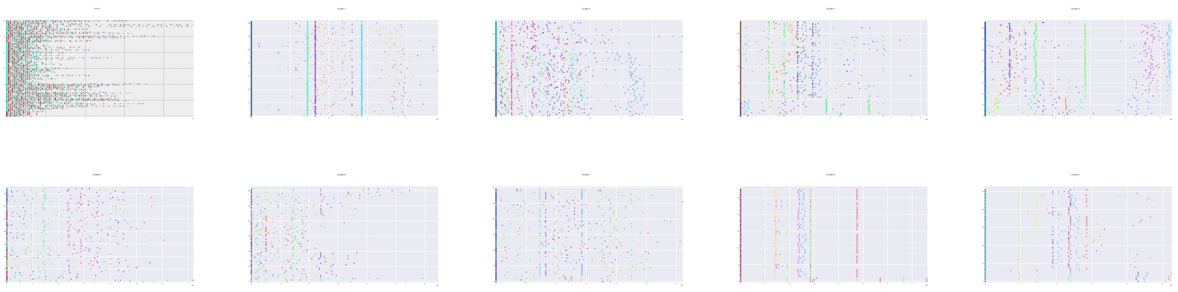


Number of MSSs



Number of TVs per sequence

First 10 clusters:



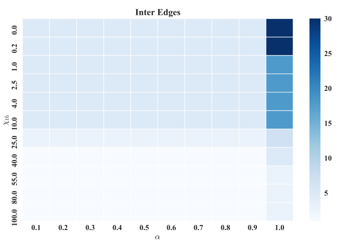
A.3.3 Specification Extraction

Next, on a subset of the segments found during Segmentation Clustering Specification Mining with BaySpec and MPE is used to find specifications. Here, only a subset of clusters and segments is considered.

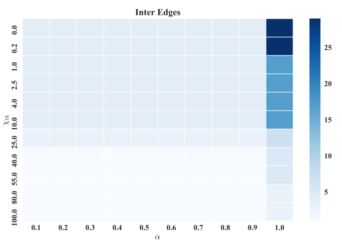
A.3.4 TV Cluster 1 - Segment Group 9

A.3.4.1 Model Training

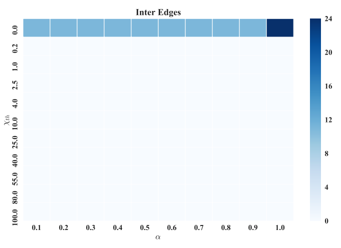
Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha = 1.10$ and $\chi_{th} = 2.5$, which gives the structure given in Figure A.128.

A.3.4.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.128. 53 specifications with a threshold $p_{min} > 0.6$ were found.
This segment resembles the automated activation of the wiper when rain is present. Sorted by likelihood the following specifications were found (the exact names of TVs were altered). The dominant specifications are the changing of the wiper position on rain intensities of b, c or d, as well as the state changes of the wiper.

- The first case includes the position going from c to b to a, on intensity of rain G(((position:c or rain:b) → X(F((position:b and X(F(position:a))))))) similar behavior occurs on intensities c and d.
- The second case includes the change of states G(((movement:leaving default position or state:wiper out of parking) → X(F(state:wiper in parking))))

The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named.
Example 1: Wiper activates on rain. Other likely MPEs vary mostly accros rain intensity values.
rain - dt=0.0 - c - dt=359624500 - b
position - dt=0.1 - c - dt=70248060 - b - dt=139869520 - a
movement - dt=0.0 - leaving default position - dt=224367702 - entering default position
state - dt=0.2 - wiper out of parking - dt=176201800 - wiper in parking

A.3.5 TV Cluster 1 - Segment Group 14

A.3.5.1 Model Training

Chosen parameters are $k = 0.00$, $\alpha = 1.0$ and $\chi_{th} = 0.2$, which gives the structure given in Figure A.129.

A.3.5.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.129. 328 specifications with a threshold $p_{min} > 0.6$ were found.
This segment resembles the manual deactivation of the wiper. Sorted by likelihood the following specifications were found (the exact names of TVs were altered). The dominant specifications are the speed of the wiper before deactivation, the deactivation trigger as well as the change of positions.

- The first case includes the final change of speed from low to high to low. G(((speed:a or speed:b) → X(G((speed:b → X(speed:a))))))
- The second case includes the deactivation. G(((control:deactivate wiper or speed:b) → X(G((speed:b → X(F(control:deactivate wiper)))))))

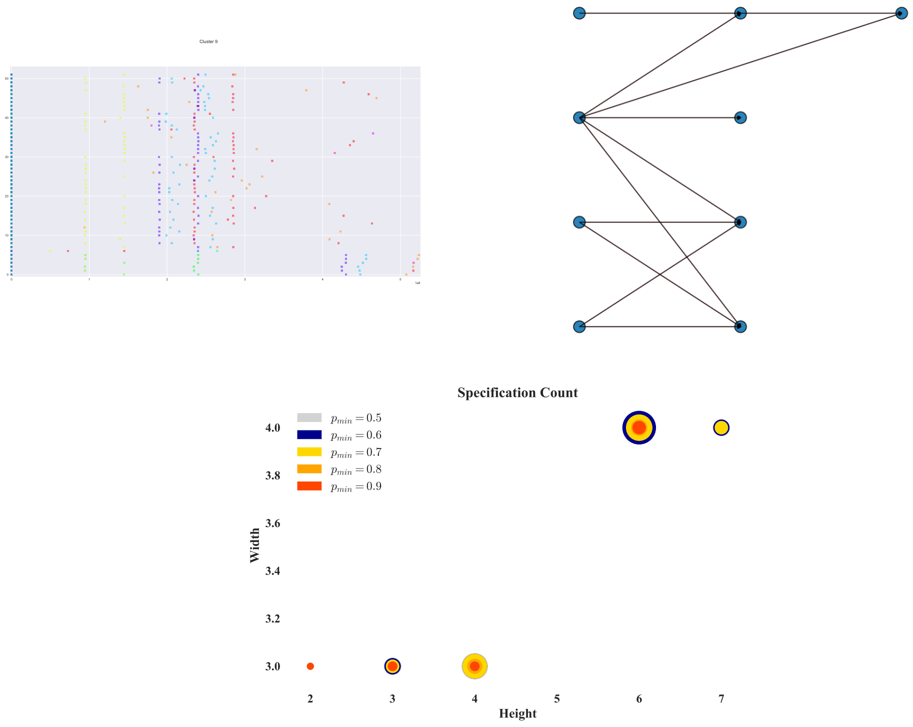


Figure A.128: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

- The third case includes cyclic movement, G(((position:d or position:a) → X(G(((position:a or position:c) → X((position:b and X((position:a and X((position:b and X(position:c)))))))))))). Here it can be seen that merging might lead to mix up of specification that makes the formula illogical, as the above specification would allow a change from d to a or from a to c in position which is not possible as the intermediate positions c, b or b are missing.

The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named.
Example 1: Wiper deactivates on user input.
speed - dt=0.0 - a - dt=141885919 - b - dt=710126295 - a TV-1 - dt=0.2 - c - dt=50097298 - d - dt=396964592 - c - dt=0.1 - c
rain - dt=0.0 - a - dt=40014638 - c - dt=114881670 - d - dt=510029596 - c
position - dt=0.0 - d - dt=46358179 - c - dt=116502348 - b - dt=114224624 - a - dt=471793609 - b - dt=114515360 - c
movement - dt=0.0 - entering default position - dt=540259555 - leaving default position
state - dt=0.0 - wiper in parking - dt=406886829 - wiper out of parking
control - dt=0.0 - deactivate wiper - dt=188515223 - b - dt=688402788 - deactivate wiper

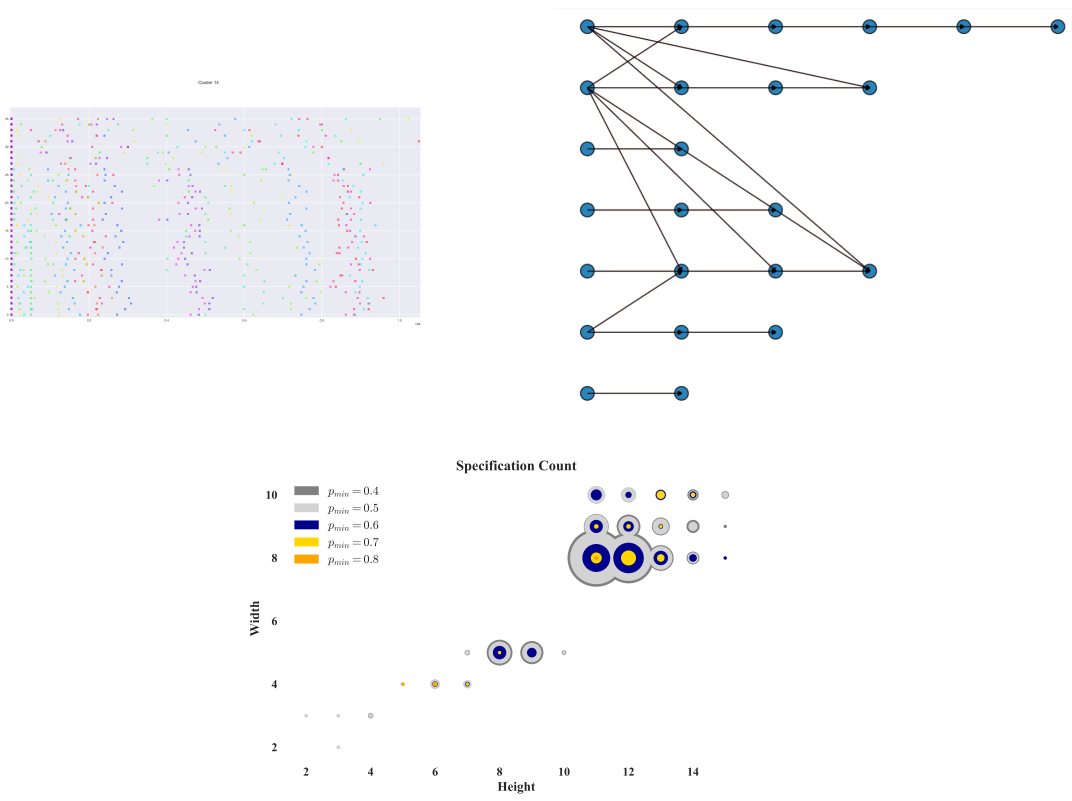


Figure A.129: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

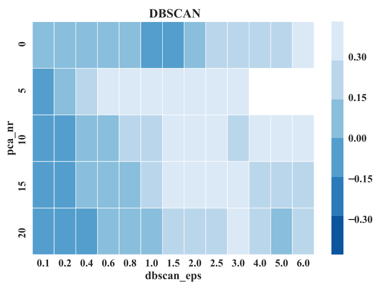
A.4 Startup

A.4.1 TV Clustering

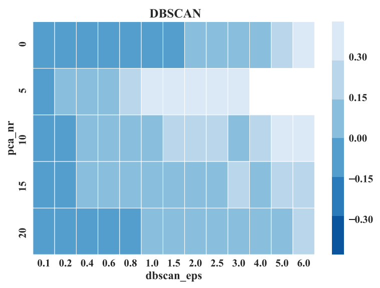
At first, TV clustering is performed with various parameters.

A.4.1.1 Hyperparameter Estimation

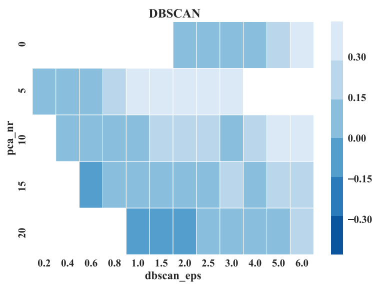
Silhouette Index, for various numbers of PCA components and various ϵ: Each plot has a fixed number set for $MinSamples = m$, when using DBSCAN



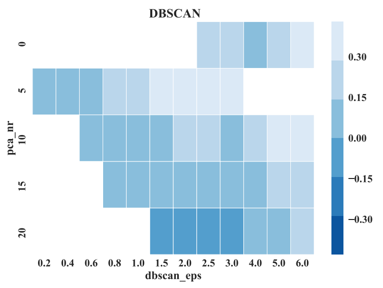
$m = 2$



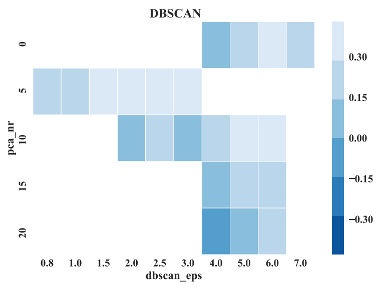
$m = 3$



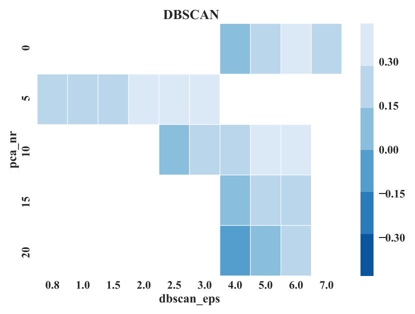
$m = 4$



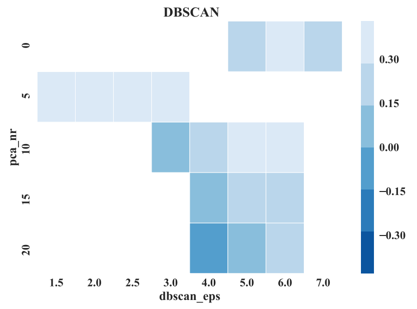
$m = 5$



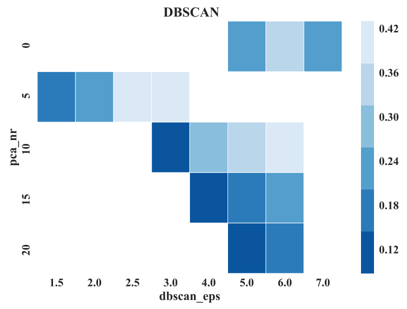
$m = 6$



$m = 7$

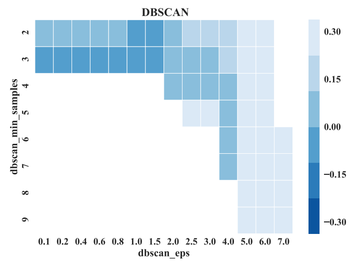


$m = 8$

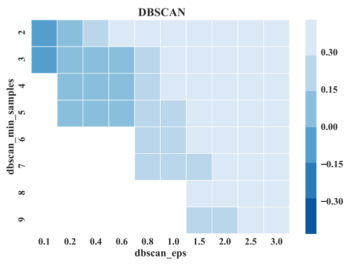


$m = 9$

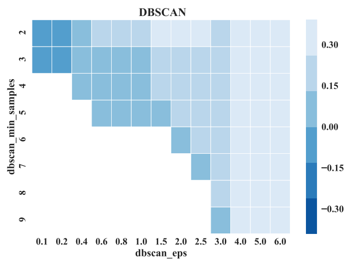
Silhouette Index, for various $MinSamples$ and various ϵ: Each plot has a fixed number set for n_{PCA}, when using DBSCAN



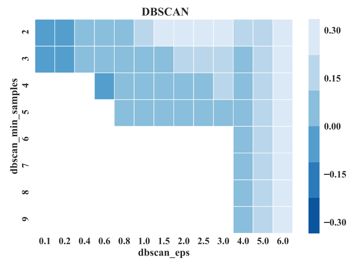
No PCA



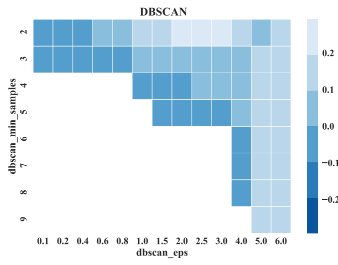
$n_{PCA} = 5$



$n_{PCA} = 10$

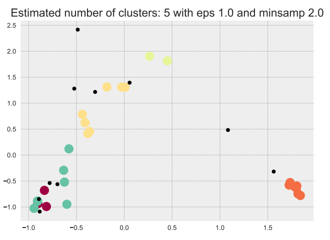


$n_{PCA} = 15$

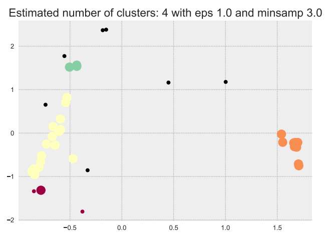


$n_{PCA} = 20$

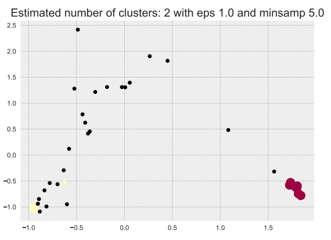
Effect of increasing $MinSamples$**:** Each plot shows the two most important PCA components. Further it has the optimal values for number of PCA and ϵ fixed, which is $n_{PCA} = 5$ and $\epsilon = 1.0$.



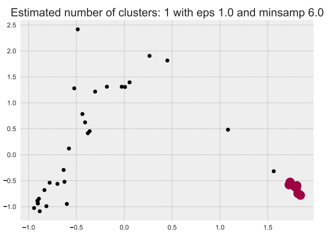
MinSamples = 2



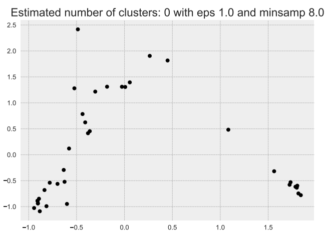
MinSamples = 3



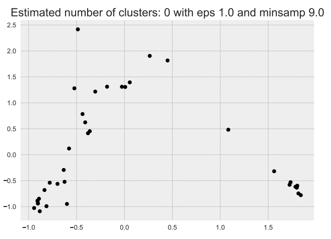
MinSamples = 5



MinSamples = 6

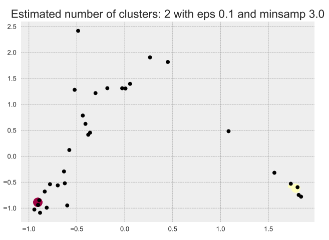


MinSamples = 8

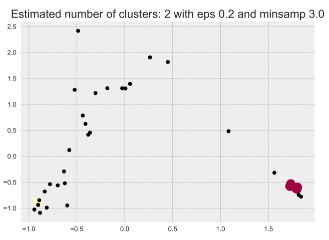


MinSamples = 9

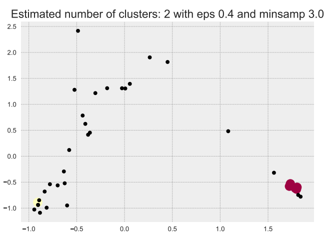
Effect of increasing ϵ**:** Each plot shows the two most important PCA components. Further it has the values for $MinSamples$ and ϵ fixed, which is $MinSamples = 3$ and $n_{PCA} = 5$.



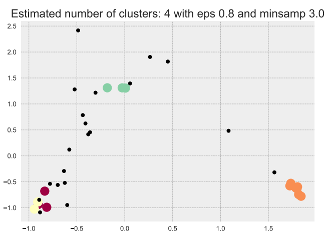
$\epsilon = 0.1$



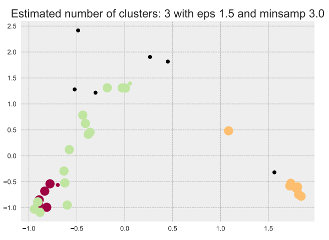
$\epsilon = 0.2$



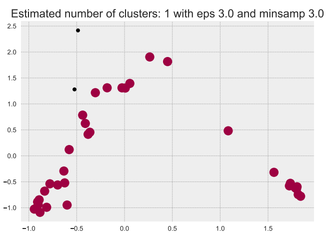
$\epsilon = 0.4$



$\epsilon = 0.8$



$\epsilon = 1.5$



$\epsilon = 3.0$

A.4.1.2 Found clusters

The algorithm found 5 clusters, including one cluster that we refer to as fragmentary which contains all TVS that were considered noise during clustering. Those clusters could be named as (1) engine and energy activation, (2) car state and energy and two clusters (3) and (4) with 3 TVs each with no meaningful description. A fragmentary cluster with (5) is also not directly nameable.

A.4.1.3 Reduced Clusters

Based on the found clustering a subset of TVs was selected by experts for further investigation.
Statistics of chosen clusters:
TV Cluster 2:

- **Number of TVs [num / nom / bin / ord]:** 32 [2/28/2/0]
- **Number of samples [num / nom / bin / ord]:** 595262 [477789/117435/38/13542]
- **Mean gap:** 10.872 s
- **Std. gap:** 12.765 s

TV Cluster 3:

- **Number of TVs [num / nom / bin / ord]:** 14 [0/13/1/0]
- **Number of samples [num / nom / bin / ord]:** 72621 [0/72592/29/0]

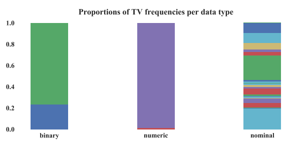


Figure A.155: TV Cluster 2: Proportions of TVs

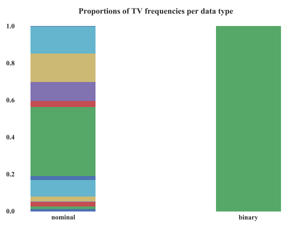


Figure A.156: TV Cluster 3: Proportions of TVs

Figure A.157: Proportion of TVs per Cluster

- **Mean gap:** 19.125 s
- **Std. gap:** 19.911 s

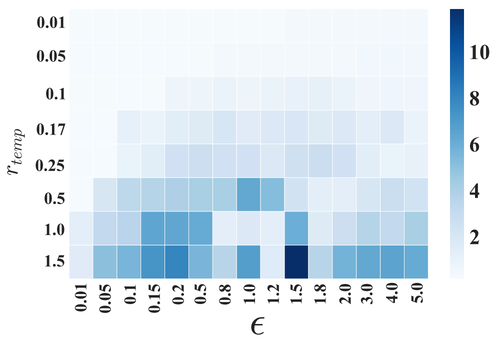
The proportion of occurrence frequencies per TV and data type is shown in Figure A.157.

A.4.2 Segmentation Clustering

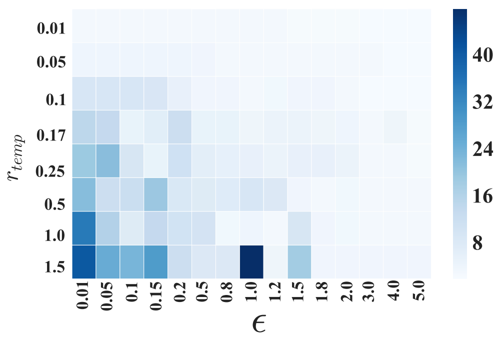
For this data set the range segmentation clustering is used. Per data set and cluster of TVs hyperparameters are found as described in this section, with statistics shown in the following.

A.4.2.1 Hyperparameters

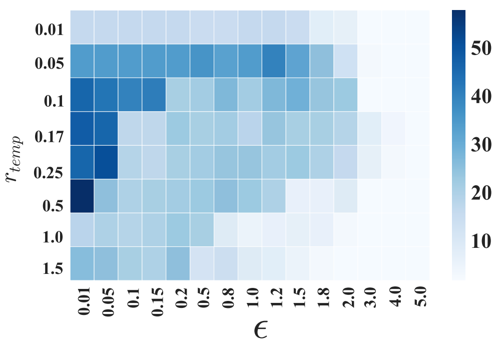
TV Cluster 2:



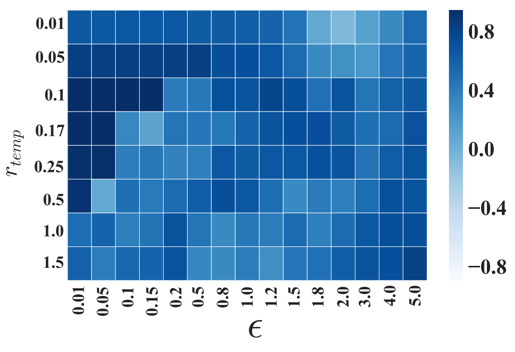
Mean of Std. of Sequence lengths per TV



Mean of Sequence lengths per TV

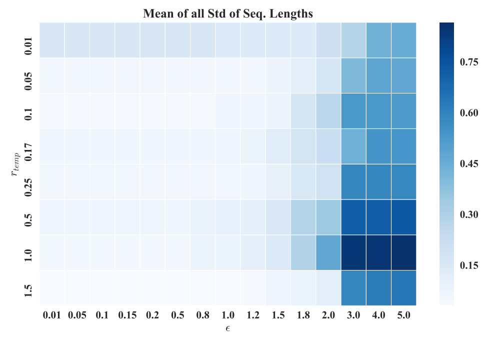


Number of Clusters found

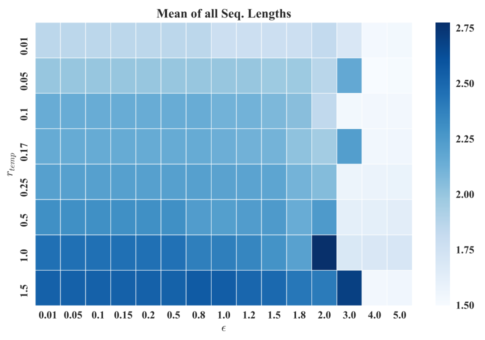


Silhouette index

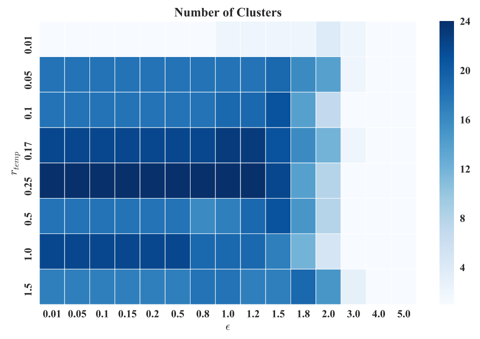
TV Cluster 3:



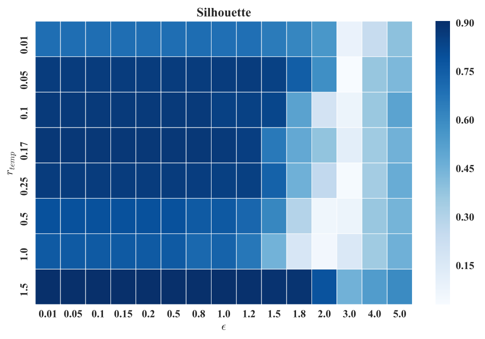
Mean of Std. of Sequence lengths per TV



Mean of Sequence lengths per TV



Number of Clusters found

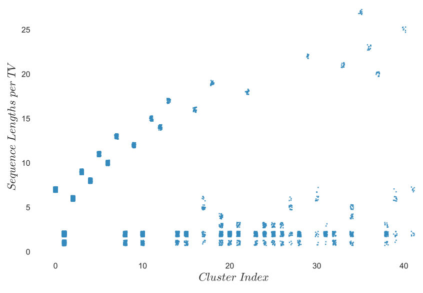


Silhouette index

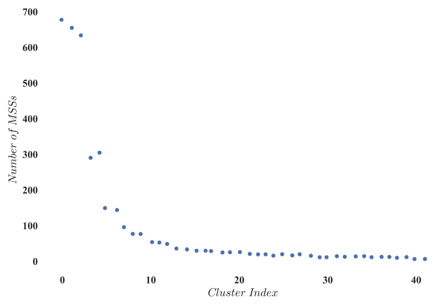
For TV cluster 1 $\epsilon = 0.01$ and $r_{temp} = 0.1$ and for TV cluster 2 $\epsilon = 0.01$ and $r_{temp} = 0.5$ were chosen.

A.4.2.2 Statistics

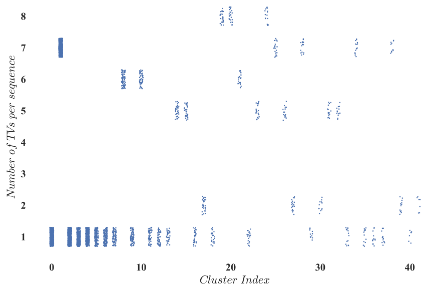
TV Cluster 2:



Sequence lengths per TV

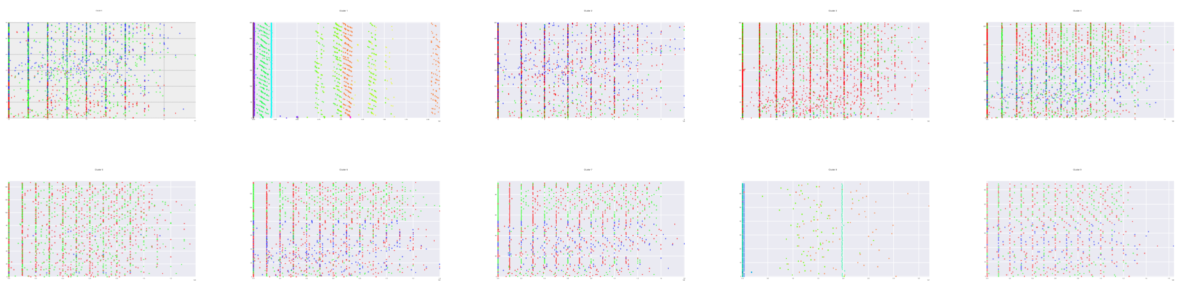


Number of MSSs

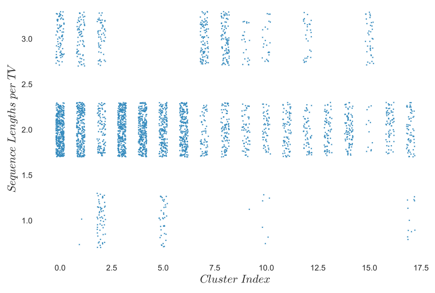


Number of TVs per sequence

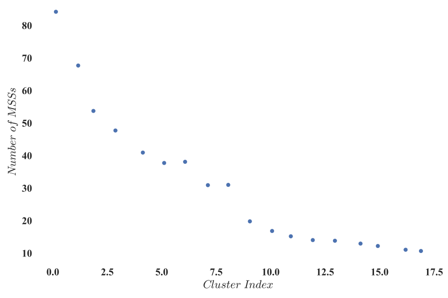
First 10 clusters:



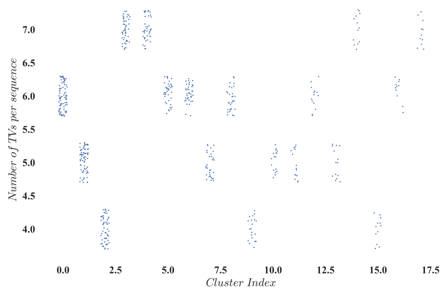
TV Cluster 3:



Sequence lengths per TV

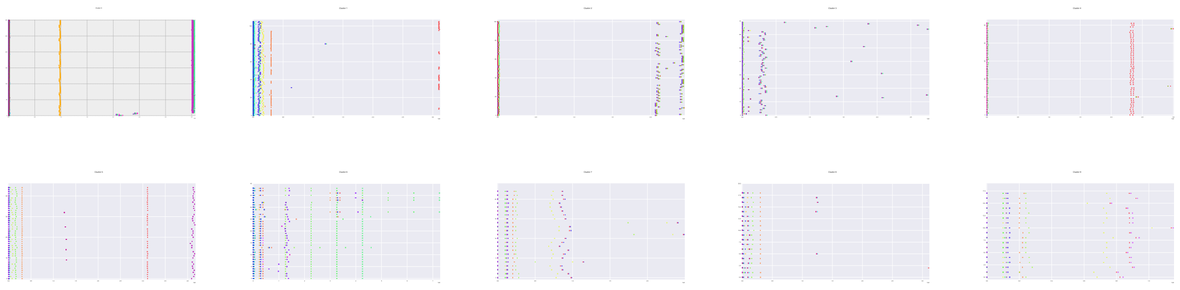


Number of MSSs



Number of TVs per sequence

First 10 clusters:



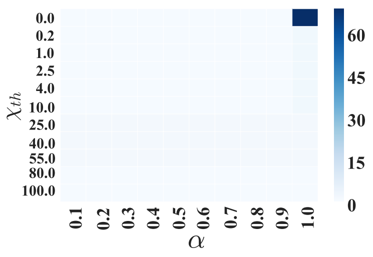
A.4.3 Specification Extraction

Next, on a subset of the segments found during Segmentation Clustering Specification Mining with BaySpec and MPE is used to find specifications. Here, only a subset of clusters and segments is considered.

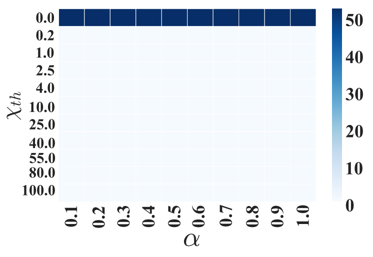
A.4.4 TV Cluster 2 - Segment Group 1

A.4.4.1 Model Training

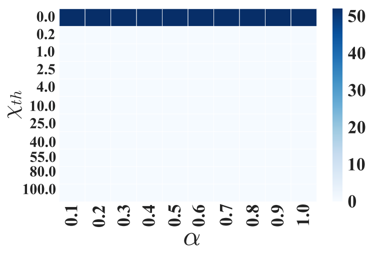
Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha = 1.0$ and $\chi_{th} = 0$, which gives the structure given in Figure A.175.

A.4.4.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.175. 1048 specifications with a threshold $p_{min} > 0.6$ were found.
This segment resembles the shutdown of multiple systems. Sorted by likelihood the following specifications were found (the exact names of TVs were altered). The dominant specifications are the shutdowns of various system components. Here, each combination of shutdowns is represented as one state of a TV. Thus, the resulting specifications, include multiple such combinations and thus, similar specifications that vary at one literal. Also, a series of gates needs to be deactivated which are named as gates within this thesis.

- An example shutdown procedure of system states is
  G(((((system-A:a off, b off, c on, d off, e on or power-supply:a) or power-supply:b) or gate-A:hold) → X(F((system-A:a off, b off, c off, d off, e off and X(F((driver:not present and X(F((system-B:f off, g on, h off, i off, j on, k off))))))))))))),
  where states were renamed.

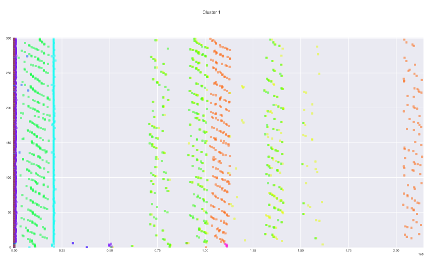
- Those combinations vary in its initial state nodes, while the consequent procedure is similar in many MSSs. BaySpec captures such intermediate procedures as well as
  G((power-supply:a → X(F(state-request:transmitted))))

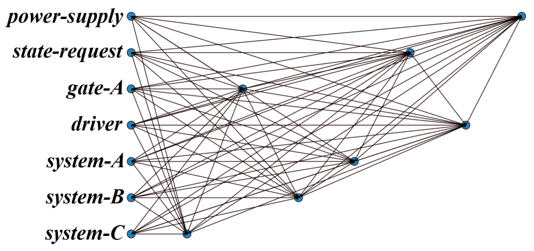
The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named.
Example 1:
power-supply - dt=0.1 - a - dt=136483490 - d
state-request - dt=0.0 - none - dt=118302256 - transmitted
gate-A - dt=0.0 - hold - dt=114115405 - off
driver - dt=0.1 - offline - dt=5504711 - not present
system-A - dt=0.0 - a off, b off, c on, d off, e on - dt=21312576 - a off, b off, c on, d off, e off
system-B - dt=0.1 - f off, g on, h off, i off, j on, k off - dt=0.3 - f off, g on, h off, i off, j off, k off
system C - dt=0.2 - a off, b off, c off , d off, e on, f on - dt=6327829 - a off, b off, c off , d off, e off, f off

A.4.5 TV Cluster 2 - Segment Group 8

A.4.5.1 Model Training

Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.





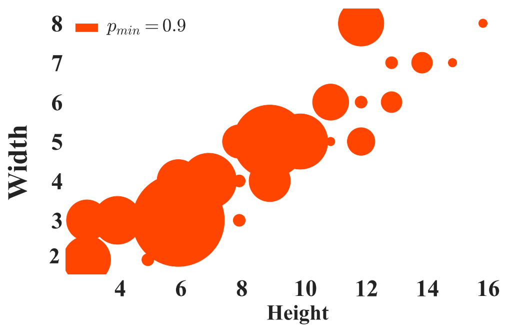
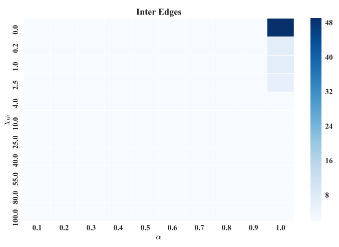
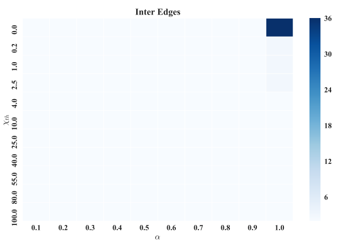


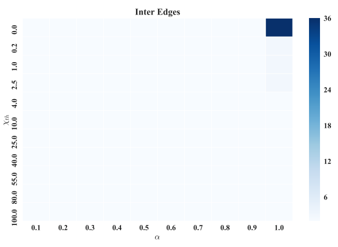
Figure A.175: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha = 1.0$ and $\chi_{th} = 0.2$, which gives the structure given in Figure A.179.

A.4.5.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.179. 36 specifications with a threshold $p_{min} > 0.6$ were found. Here, the shutdown procedure in terms of subsystems is resembled. Similar to the case of TV Cluster 1 and group 1, the specifications capture variations of initial TV nodes.

- The dominant pattern is of shape G(((subsystem-A:offline or system-A:a off, b off, c of, d off, e off) $\rightarrow$ X(F((system-B:f off, g off, h off, i off, j off and X(F(subsystem control:system x off, system y off, system z off)))))))

The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named. The three dominant state changes include, the process of the user leaving the car, the deactivation of systems and deactivation of the infrastructure.
Example 1:
state-request - dt=0.0 - transmitted - dt=29902476 - transmitted
driver - dt=0.0 - not present - dt=13647222 - offline
subsystem control - dt=0.0 - system x off, system y on, system z off - dt=13229846 - system x off, system y off, system z off
system-A - dt=0.0 - a on, b off, c on, d off, e on - dt=18276993 - a off, b off, c off, d off, e on
system-B - dt=0.0 - f off, g on, h off, i off, j off, k off - dt=0.2 - f off, g on, h off, i off, j off, k off
system-C - dt=0.0 - a off, b off, c off , d off, e on, f on - dt=285766 - a off, b off, c off , d off, e off, f off

A.4.6 TV Cluster 2 - Segment Group 15

A.4.6.1 Model Training

Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.

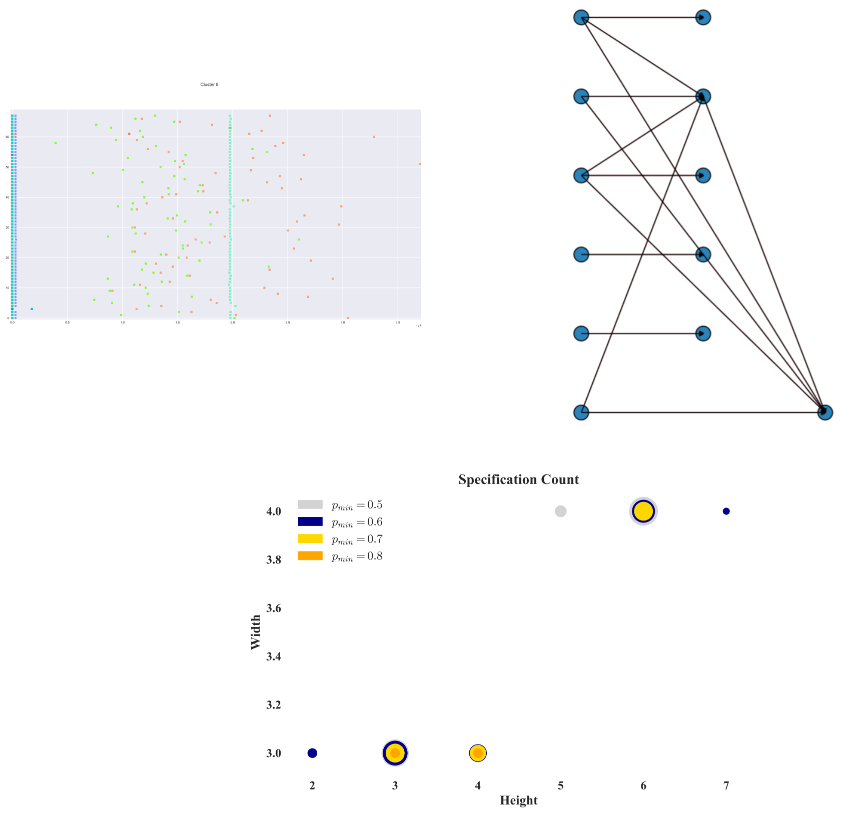
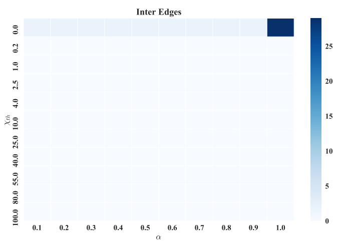
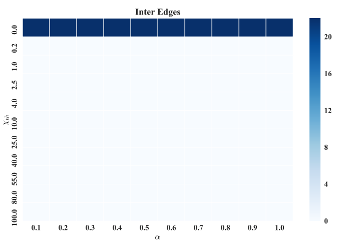


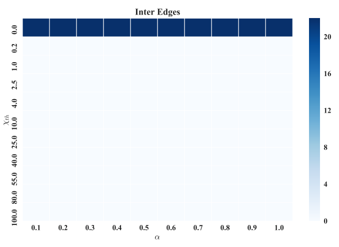
Figure A.179: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha = 0.9$ and $\chi_{th} = 0.0$, which gives the structure given in Figure A.183.

A.4.6.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.183. 4 specifications with a threshold $p_{min} > 0.6$ were found.
Here, the start procedure in terms of engine activation is represented. Found specifications are short and restricted to two TVs here, which results from a structure with less connections.

- The dominant pattern is the start of the engine when connected to the system state activations.
  G(((system-A:a on, b on, c on, d on, e on or engine:started) $\rightarrow$ X(F(system-B:f on, g on, h on, i on))))

Here, only four variations of this patter were found.

The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named. Dominant states are the engine activation here.
Example 1:
state - dt=0.1 - initialize - dt=683683 - driving initialized
engine - dt=0.0 - starting - dt=0.2 - on
driver - dt=0.0 - starting - dt=4141373 - driving
subsystem-A - dt=0.0 - initialize - dt=21320401 - driving initialized
system-A - dt=0.0 - a on, b on, c on, d on, e on

A.4.7 TV Cluster 3 - Segment Group 0

A.4.7.1 Model Training

Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.

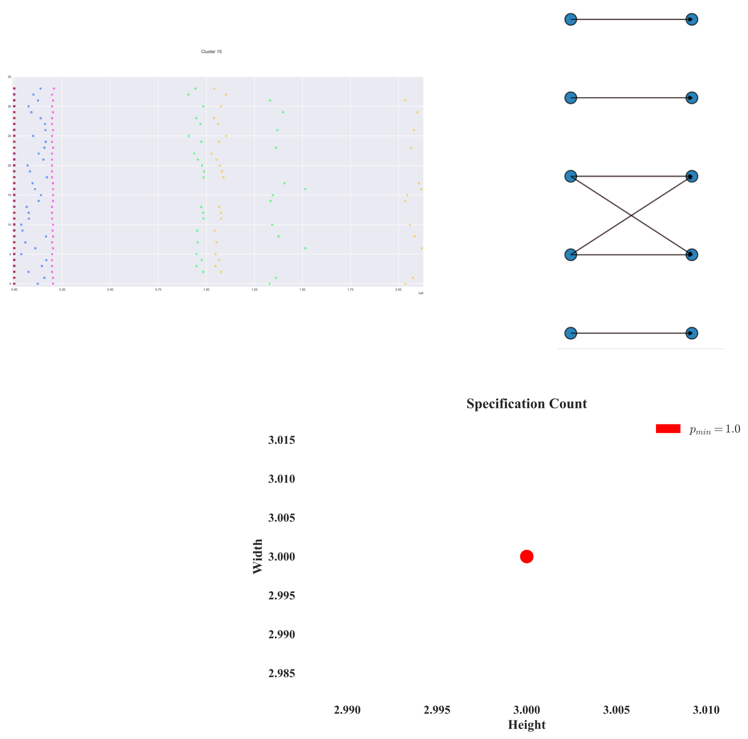
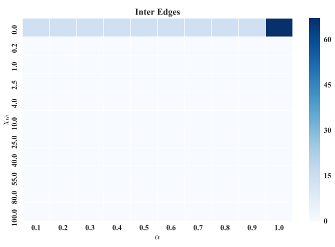
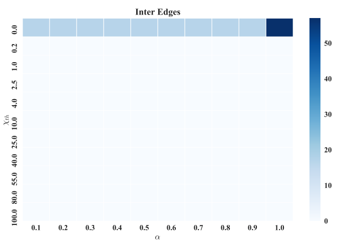


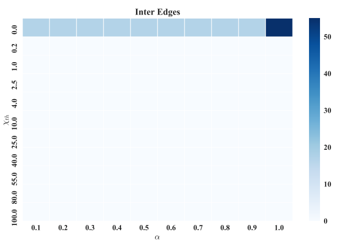
Figure A.183: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha = 0.9$ and $\chi_{th} = 0.0$, which gives the structure given in Figure A.187.

A.4.7.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.187. 88 specifications with a threshold $p_{min} > 0.6$ were found. Here, the shutdown procedure in terms of engine activation, subsystems and system activations is represented.

- First, this includes the subsystem deactivation and vehicle states such as G((subsystem-A:deactivating → X(F((subsystem-A:offline or driver:not present)))))
- Second, this includes combinations of system deactivation and subsystems. G(((subsystem control:system x off, system y off, system z off, w off or subsystem-A:deactivating) → X(F(subsystem control:system x off, system y on, system z off, w off))))
- Third, similar to the above case the main sequence is found that is followed by the initial variations. G((subsystem-A:deactivating → X(F((subsystem-A:offline or driver:not present)))))

The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named. .
Example 1:
gate-A - dt=0.3 - hold - dt=55011845 - hold
driver - dt=0.0 - offline - dt=8075831 - not present
subsystem control - dt=0.0 - system x off, system y off, system z off dt=115710907 - system x off, system y off, system z off
system-A - dt=0.0 - a off, b off, c off, d off, e on - dt=11980516 - a on, b off, c off, d off, e on
system-B - dt=0.1 - f off, g on, h off, i off, j off, k off - dt=81573110. - f off, g on, h on, i on, j off, k off
system-C - dt=0.2 - a off, b off, c off , d off, e off, f off - dt=0.0 - a on, b on, c on , d off, e on, f on - dt=47271014 - a on, b on, c on , d off, e on, f on

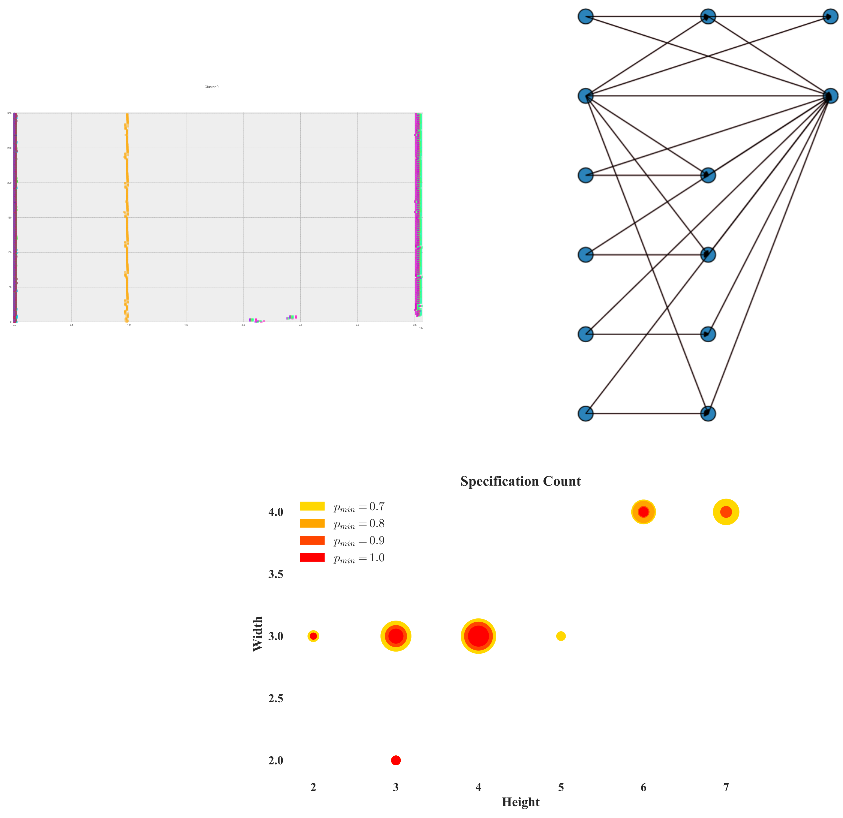


Figure A.187: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

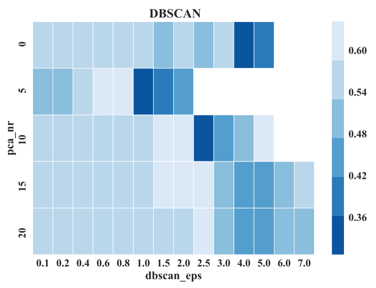
A.5 Shutdown

A.5.1 TV Clustering

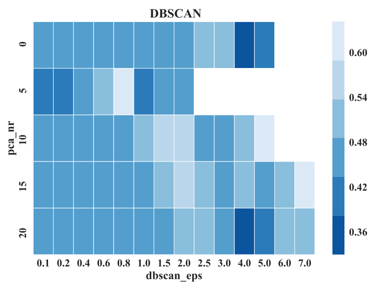
At first, TV clustering is performed with various parameters.

A.5.1.1 Hyperparameter Estimation

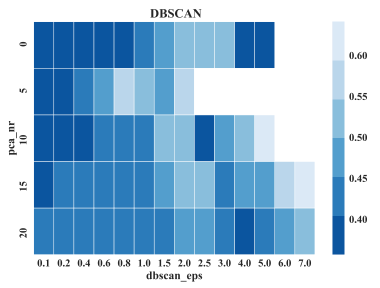
Silhouette Index, for various numbers of PCA components and various ϵ: Each plot has a fixed number set for $MinSamples = m$, when using DBSCAN



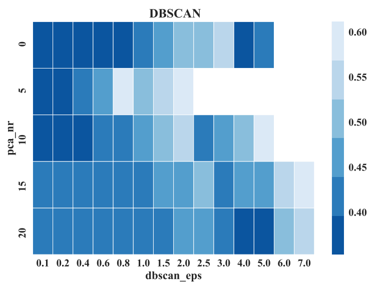
$m = 2$



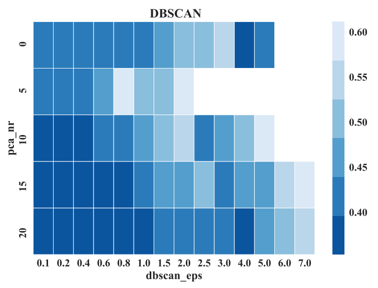
$m = 3$



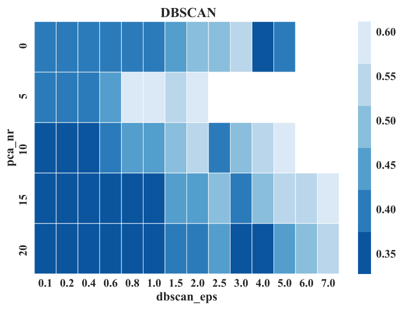
$m = 4$



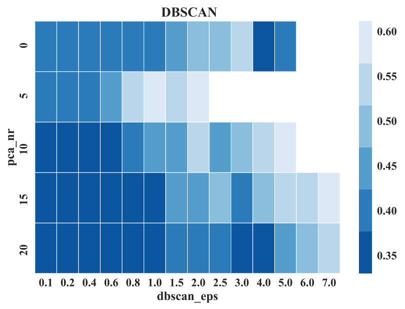
$m = 5$



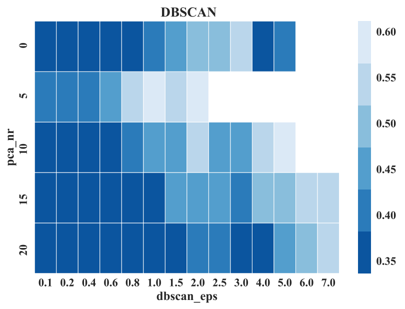
$m = 6$



$m = 7$

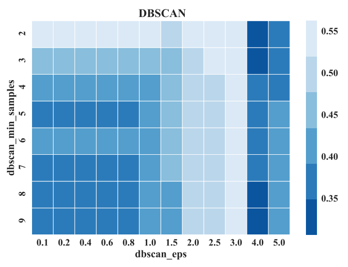


$m = 8$

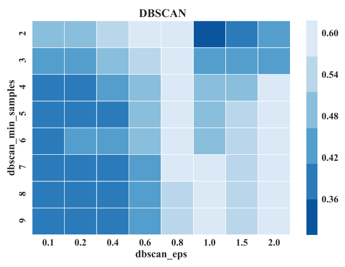


$m = 9$

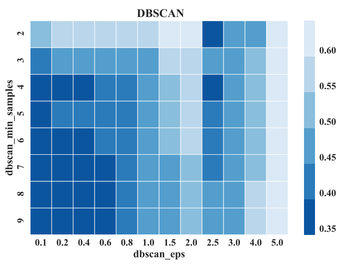
Silhouette Index, for various $MinSamples$ **and various ϵ:** Each plot has a fixed number set for n_{PCA}, when using DBSCAN



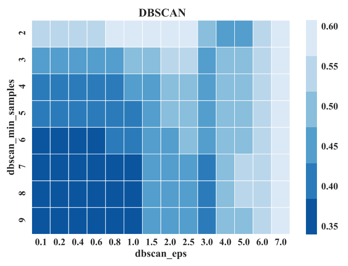
No PCA



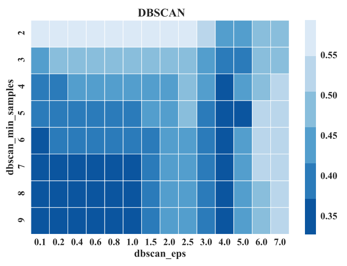
$n_{PCA} = 5$



$n_{PCA} = 10$

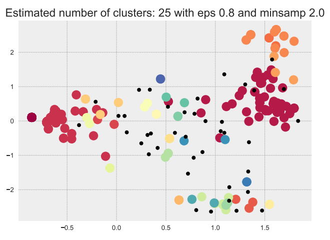


$n_{PCA} = 15$

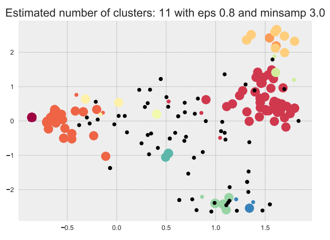


$n_{PCA} = 20$

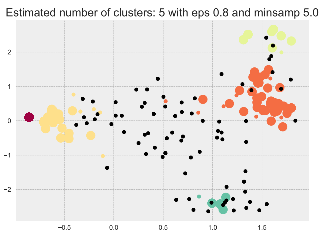
Effect of increasing $MinSamples$**:** Each plot shows the two most important PCA components. Further it has the optimal values for number of PCA and ϵ fixed, which is $n_{PCA} = 5$ and $\epsilon = 0.8$.



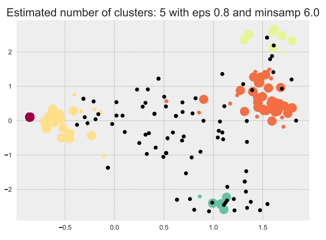
MinSamples = 2



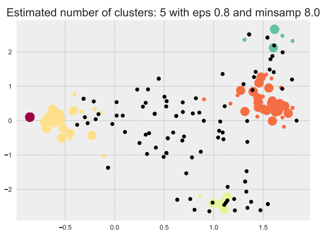
MinSamples = 3



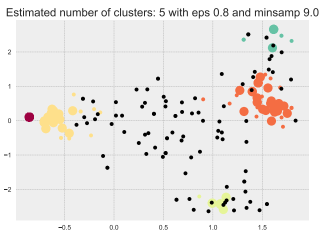
MinSamples = 5



MinSamples = 6

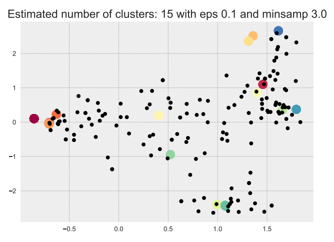


MinSamples = 8

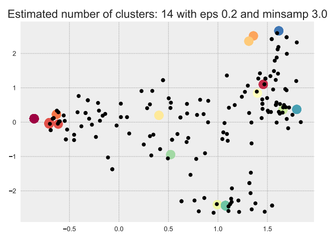


MinSamples = 9

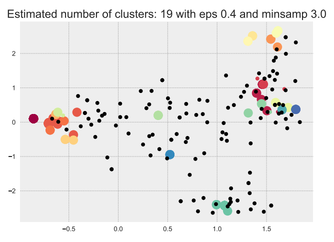
Effect of increasing ϵ**:** Each plot shows the two most important PCA components. Further it has the values for $MinSamples$ and ϵ fixed, which is $MinSamples = 3$ and $n_{PCA} = 5$.



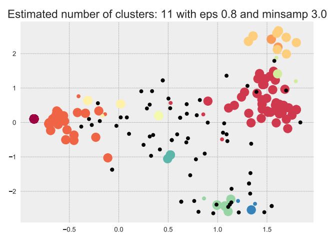
$\epsilon = 0.1$



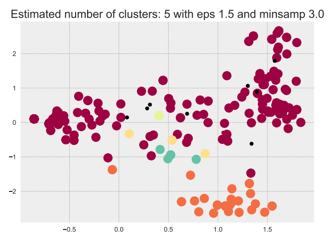
$\epsilon = 0.2$



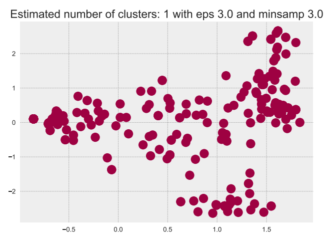
$\epsilon = 0.4$



$\epsilon = 0.8$



$\epsilon = 1.5$



$\epsilon = 3.0$

A.5.1.2 Found clusters

The algorithm found 11 clusters, including one cluster that we refer to as fragmentary which contains all TVS that were considered noise during clustering. Most dominant clusters could be named as (1) car state information before shut down, (2) state information post shutdown or (3) key information. Due to the big size of this data set a subclustering is possible, which is omitted here, as a similar case was already studied in the main work.

A.5.1.3 Reduced Clusters

Based on the found clustering a subset of TVs was selected by experts for further investigation.
Statistics of chosen clusters:
TV Cluster 1:

- **Number of TVs [num / nom / bin / ord]:** 24 [0/18/6/0]
- **Number of samples [num / nom / bin / ord]:** 1569 [0/1451/118/0]
- **Mean gap:** 8.223 s
- **Std. gap:** 22.098 s

TV Cluster 2:

- **Number of TVs [num / nom / bin / ord]:** 7 [0/7/0/0]
- **Number of samples [num / nom / bin / ord]:** 3505 [0/3505/0/0]

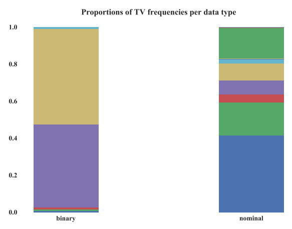


Figure A.213: TV Cluster 1: Proportions of TVs

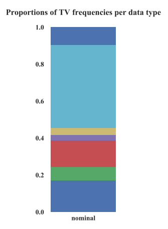


Figure A.214: TV Cluster 2: Proportions of TVs

Figure A.215: Proportion of TVs per Cluster

- **Mean gap:** 34.893 s
- **Std. gap:** 10.248 s

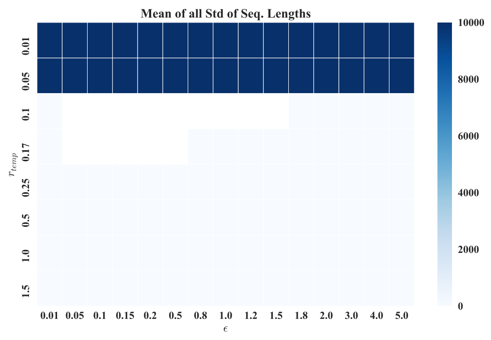
The proportion of occurrence frequencies per TV and data type is shown in Figure A.215.

A.5.2 Segmentation Clustering

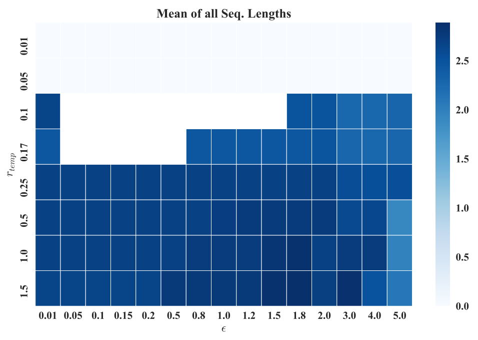
For this data set the range segmentation clustering is used. Per data set and cluster of TVs hyperparameters are found as described in this section, with statistics shown in the following.

A.5.2.1 Hyperparameters

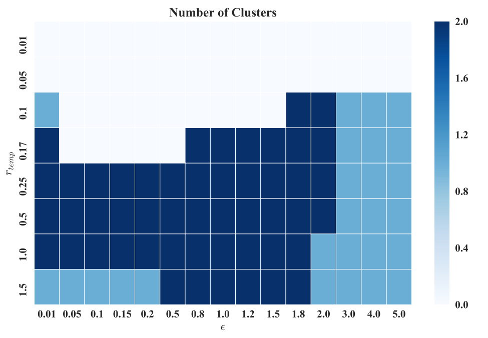
TV Cluster 1:



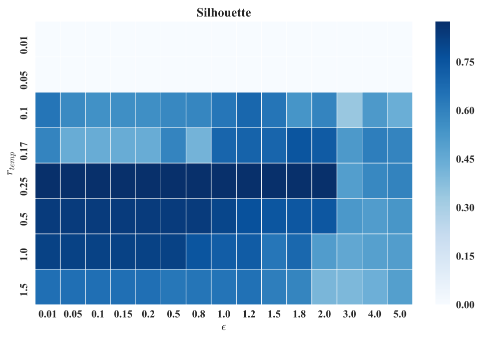
Mean of Std. of Sequence lengths per TV



Mean of Sequence lengths per TV

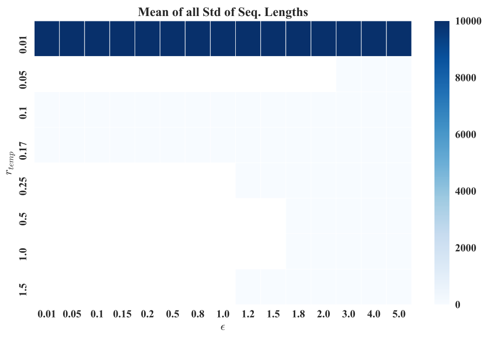


Number of Clusters found

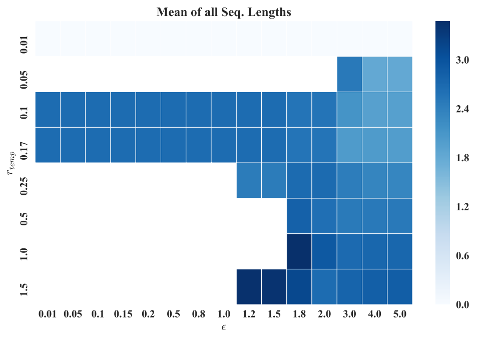


Silhouette index

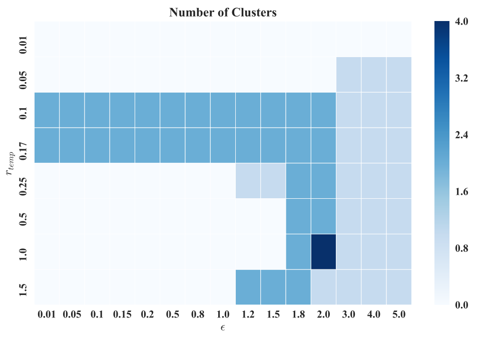
TV Cluster 2:



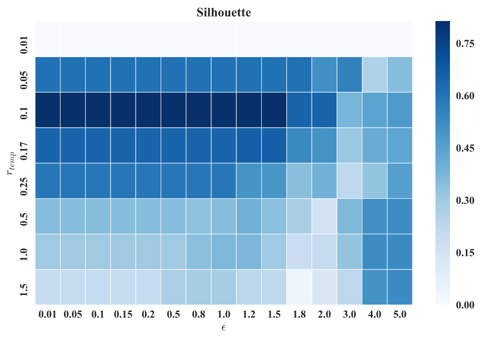
Mean of Std. of Sequence lengths per TV



Mean of Sequence lengths per TV



Number of Clusters found

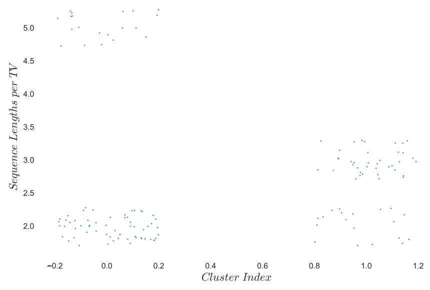


Silhouette index

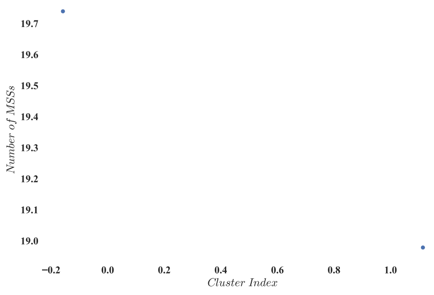
For TV cluster 1 $\epsilon = 1.0$ and $r_{temp} = 0.25$ and for TV cluster 2 $\epsilon = 0.01$ and $r_{temp} = 0.1$ were chosen.

A.5.2.2 Statistics

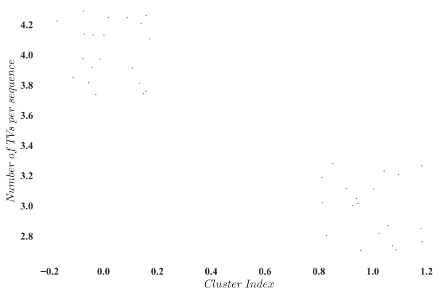
TV Cluster 1:



Sequence lengths per TV

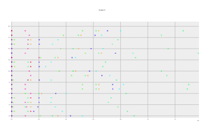


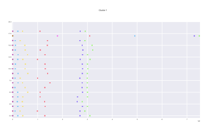
Number of MSSs



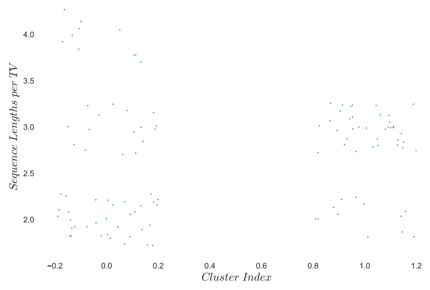
Number of TVs per sequence

Two found clusters:

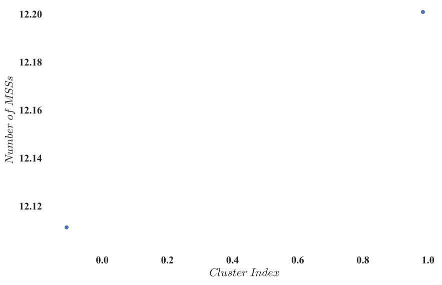




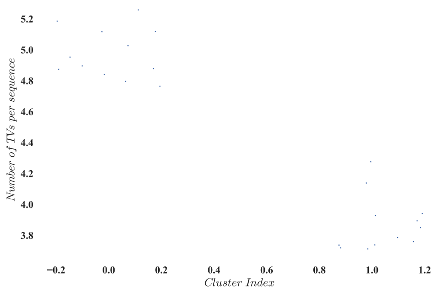
TV Cluster 2:



Sequence lengths per TV

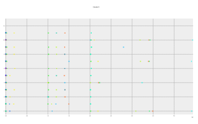
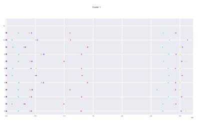


Number of MSSs



Number of TVs per sequence

Two found clusters:

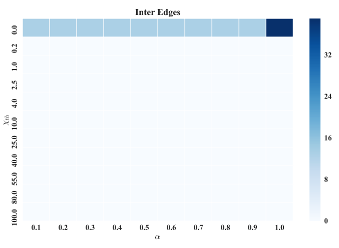
A.5.3 Specification Extraction

Next, on a subset of the segments found during Segmentation Clustering Specification Mining with BaySpec and MPE is used to find specifications. Here, only a subset of clusters and segments is considered.

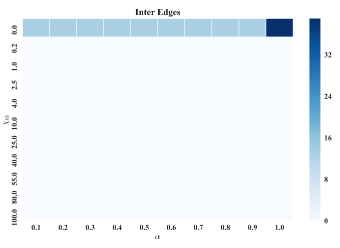
A.5.4 TV Cluster 1 - Segment Group 0

A.5.4.1 Model Training

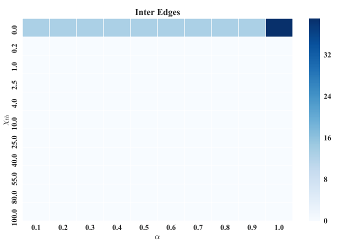
Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha =$ and $\chi_{th} =$, which gives the structure given in Figure A.233.

A.5.4.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.233. 209 specifications with a threshold $p_{min} > 0.6$ were found.
This segment resembles the start up of multiple systems. Sorted by likelihood the following specifications were found (the exact names of TVs were altered). The dominant specifications are the vehicle state, engine component activations and the start button.

- First, the main patter includes the component activations. G(((driver:present or engine-component:a on) → X(F((engine-component:b changing and X((engine-component:b on and X(F(driver:driving))))))))) or the same situation when the driver was in another state G(((driver:in parking or engine-component:a on) → X(F((engine-component:b changing and X((engine-component:b on and X(F(driver:driving)))))))))
- The above behavior was also found when including the start button. G(((driver:present or start button:activation) → X(F(driver:driving))))
- Combination of all TVs. G(((((engine-component:a on or driver:in parking) or driver:present) or driver:driving) → X(F((engine-component:b changing and X((engine-component:b on and X((engine-component:c on and X(F(engine-component:b on)))))))))))

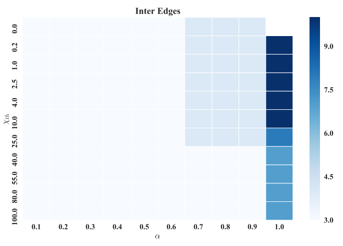
The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named.

Example 1:
engine - dt=0.0 - off - dt=206202769 - starting
engine-component - dt=0.0 - a on - dt=71543944 - b changing - dt=97400191 - b on - dt=210268894 - c on - dt=466739935 - b on
driver - dt=0.5 - in parking - dt=67200404 - driving
start button - dt=0.0 - activation - dt=112656636 - no action

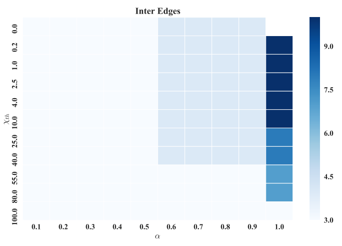
A.5.5 TV Cluster 2 - Segment Group 0

A.5.5.1 Model Training

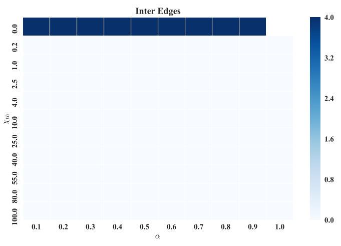
Results of Hyperparameter evaluation in terms of inter edges when varying α and χ_{th}.



$k = 0.00$



$k = 0.15$



$k = 0.35$

Chosen parameters are $k = 0.00$, $\alpha =$ and $\chi_{th} =$, which gives the structure given in Figure A.237.

A.5.5.2 Specification Extraction

With this BaySpec and MPE are used to produce specifications as shown in Figure A.237. 503 specifications with a threshold $p_{min} > 0.6$ were found. This segment resembles the start up of multiple systems. The structure has less connections giving specifications among three TVs only. The dominant specifications are the driver behavior, the state transitions after pressing the start button and various reasons to change from parking to driving.

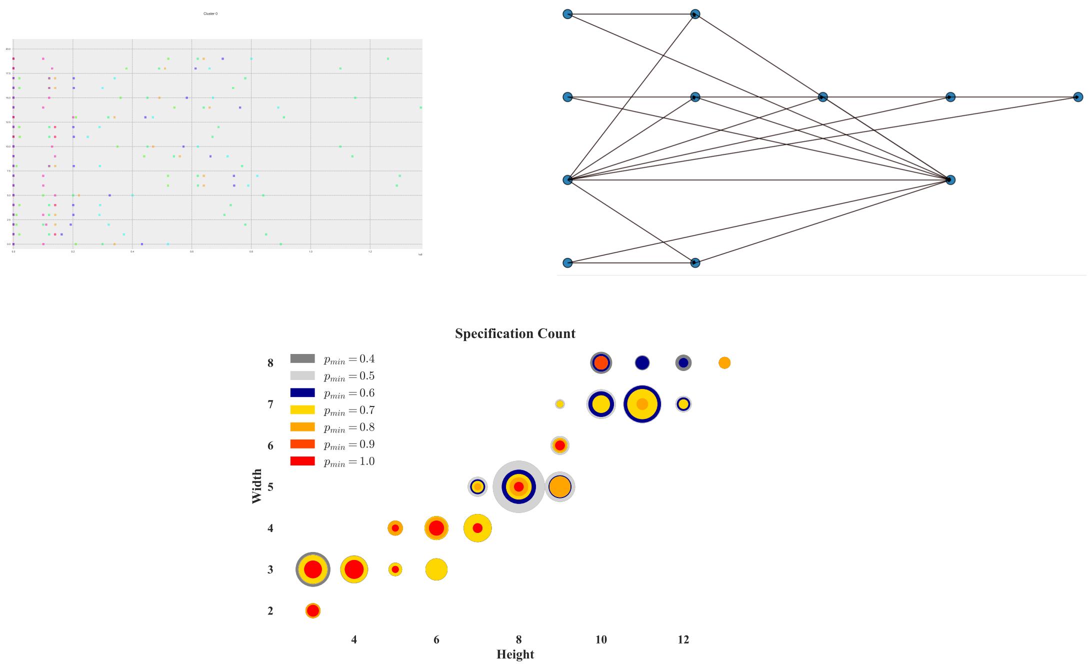


Figure A.233: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

- First, the main patter includes the driver behavior across one TV only.
  G(((driver:present or driver:initialized driving) $\rightarrow$ X(driver:driving)))
- Second, the sequence after pressing the start button.
  G((start button:activated $\rightarrow$ X(F((trigger state:gate activated and X(activation trigger state:driver intends activation))))))
- Third, a combination of both.
  G(((driver:present or start button:activated) $\rightarrow$ X(F((trigger state:gate activated and X(activation trigger state:driver intends activation))))))

The dominating MPEs are given in the following where dt is specified in nano seconds and a subset of TVs is named.
Example 1:
engine - dt=0.1 - off - dt=127023990 - starting
engine-component - dt=0.0 - a on - dt=0.0 - b changing - dt=103681666 - b on
activation trigger state - dt=0.1 - driver intends activation - dt=1459612 - driver activates using start button - dt=118070992 - activation by b on - dt=96969978 - driving initialized
driver - dt=0.1 - in parking - dt=457282028 - driving
start button - dt=0.0 - activation - dt=87760836 - no action - dt=0.0 - no action

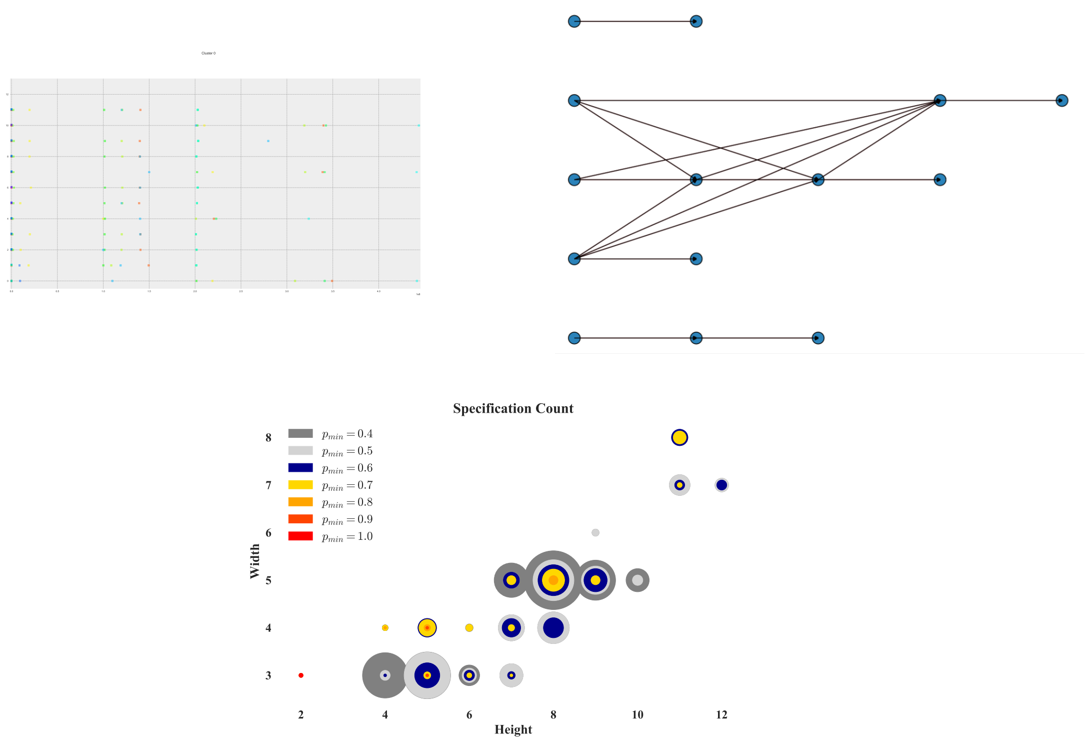


Figure A.237: Flow of extraction. On the left the MSS segments used for training are given. The middle shows the learned structure and the right side shows the complexity of found specifications. Size of circles indicates the number of found specifications with this complexity that have likelihood bigger than p_{min}.

B Appendix B: Deriving Update Equation

In Chapter 7 the update equation for the temporal estimate was presented. Here, a deduction of this equation is given.

First, it is known that

$$\begin{aligned} q^*(\Delta t_{ij}; \mu^*, {\sigma^*}^2) &\propto \mathbb{E}_{\Delta t_{-ij}}[\log p(Z, X)] \\ \propto \sum_{\omega \in Pa(\Delta t_{ij})} \log p(\Delta t_{ij}; \mu_{ij}, {\sigma_{ij}}^2 | Pa(\Delta t_{ij}) = \omega) &\cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r) \\ \propto \sum_{\omega \in Pa(\Delta t_{ij})} \log \mathcal{N}(\mu_{ij|\omega}, \sigma^2) &\cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r) \end{aligned}$$

, with

$$\mathcal{N}(\mu_{ij|\omega}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp(-\frac{(\Delta t_{ij|\omega} - \mu_{ij|\omega})^2}{2\sigma^2})$$

Thus,

$$\begin{aligned} q^*(\Delta t_{ij}; \mu_{ij}^*, {\sigma_{ij}^*}^2) &\propto \exp(\sum_{\omega \in Pa(\Delta t_{ij})} \log \mathcal{N}(\mu_{ij|\omega}, \sigma^2) \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)) \\ \propto \exp(\sum_{\omega \in Pa(\Delta t_{ij})} &\log(\frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp(-\frac{(\Delta t_{ij|\omega} - \mu_{ij|\omega})^2}{2\sigma^2})) \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)) \\ \propto \exp(\sum_{\omega \in Pa(\Delta t_{ij})} &(\log(\frac{1}{\sqrt{2\pi\sigma^2}}) + \log(\exp(-\frac{(\Delta t_{ij|\omega} - \mu_{ij|\omega})^2}{2\sigma^2}))) \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)) \\ \propto \exp(\sum_{\omega \in Pa(\Delta t_{ij})} &(\log(\frac{1}{\sqrt{2\pi\sigma^2}}) - \frac{(\Delta t_{ij|\omega} - \mu_{ij|\omega})^2}{2\sigma^2}) \cdot \prod_{\omega_r \in \omega} q(v_r = \omega_r)) \end{aligned}$$

with

$$\begin{aligned} a_1 &= \log(\frac{1}{\sqrt{2\pi\sigma^2}} \\ a_2 &= \frac{1}{2\sigma^2} \\ a_3(\omega) &= \prod_{\omega_r \in \omega} q(v_r = \omega_r)) \end{aligned}$$

the above can be written as

$$q^*(\Delta t_{ij}; \mu^*_{ij}, {\sigma^*_{ij}}^2) \propto \exp(\sum_{\omega \in Pa(\Delta t_{ij})} (a_1 - a_2(\Delta t_{ij|\omega} - \mu_{ij|\omega})^2) \cdot a_3(\omega))$$

$$\propto \exp(\sum_{\omega \in Pa(\Delta t_{ij})} (-a_2 \cdot a_3(\omega) \Delta t^2_{ij|\omega} + (-2 \cdot a_3(\omega) a_2 \mu_{ij|\omega}) \Delta t_{ij|\omega} + (\mu^2_{ij|\omega} a_2 \cdot a_3(\omega) + a_1))$$

, which can be used to read of parameters of a normal distribution as it is of shape $e^{(ax^2+bx+c)}$. That is,

$$a = \sum_{\omega \in Pa(\Delta t_{ij})} -a_2 \cdot a_3(\omega)$$

$$b = \sum_{\omega \in Pa(\Delta t_{ij})} (-2a_2 \cdot a_3(\omega) \mu_{ij|\omega})$$

$$c = \sum_{\omega \in Pa(\Delta t_{ij})} (\mu^2_{ij|\omega} a_2 \cdot a_3(\omega) + a_1)$$

With this,

$$q^*(\Delta t_{ij}; \mu^*_{ij}, {\sigma^*_{ij}}^2) \propto \exp\left(a \Delta t^2_{ij|\omega} + b \Delta t_{ij|\omega} + c\right)$$

and thus,

$$\mu^*_{ij} = -\frac{b}{2a} = -\frac{\sum\limits_{\omega \in Pa(\Delta t_{ij})} (-2a_2 \cdot a_3(\omega) \mu_{ij|\omega})}{2(\sum\limits_{\omega \in Pa(\Delta t_{ij})} -a_2 \cdot a_3(\omega))}$$

$$= \frac{\sum\limits_{\omega \in Pa(\Delta t_{ij})} (2a_2 \cdot a_3(\omega) \mu_{ij|\omega})}{2(\sum\limits_{\omega \in Pa(\Delta t_{ij})} a_2 \cdot a_3(\omega))} \text{ and as } a_2 \text{ is constant here}$$

$$= \frac{\sum\limits_{\omega \in Pa(\Delta t_{ij})} a_3(\omega) \mu_{ij|\omega}}{\sum\limits_{\omega \in Pa(\Delta t_{ij})} a_3(\omega)}$$

$$\mu^*_{ij} = \frac{\sum\limits_{\omega \in Pa(\Delta t_{ij})} \mu_{ij|\omega} \prod_{\omega_r \in \omega} q(v_r = \omega_r)}{\sum\limits_{\omega \in Pa(\Delta t_{ij})} \prod_{\omega_r \in \omega} q(v_r = \omega_r))}$$

Note that here σ was assumed constant.