UNIVERSITY OF CALIFORNIA,
IRVINE

A Framework for Discovering Implicit Knowledge from Event Logs

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computational Science

by

Zhenyu Zhang

2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

First and foremost I am extremely grateful to my thesis advisor Prof. Shangping Ren and co-advisor Prof. Nalini Venkatasubramanian for their invaluable advice, continuous support, and patience during my Ph.D. study. Their immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. I would also like to thank my former team members, Dr. Chunhui Guo, Dr. Zhicheng Fu, and Dr. Xiayu Hua, for their technical support on my study. I would like to thank all the team members from the University of California-Irvine and San Diego State University. It is their kind help and support that have made my study and life in the USA a wonderful time. I would also like to thank Prof. Sharad Mehrotra, Prof. Alex Nicolau, and Prof. Wei Wang for being in my thesis committee and for their helpful suggestions and advice. I thank my wife, Mrs. Rui Xu, for her love and support throughout my Ph.D. program. Finally, I would like to express my gratitude to my parents, Mr. Lin Zhang and Mrs. Guiying Wei. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

Special thanks to the National Science Foundation (NSF) for the support to my research work under grants NSF 1952247, NSF 1952225, and NSF 1929469.

VITA

Zhenyu Zhang

EDUCATION

Doctor of Philosophy in Computational Science	**2022**
University of California, Irvine	*Irvine, California*
San Diego State University, San Diego	*San Diego, California*
Master of Software Engineering	**2014**
University of Wisconsin, La Crosse	*La Crosse, Wisconsin*
Bachelor of Science in Computer Sciences	**2012**
Wuhan University	*Wuhan, China*

RESEARCH EXPERIENCE

Graduate Research Assistant	**2021–2022**
University of California, Irvine	*Irvine, California*
Graduate Research Assistant	**2017–2021**
San Diego State University	*San Diego, California*
Graduate Research Assistant	**2015–2016**
Illinois Institute of Technology	*Chicago, Illinois*

TEACHING EXPERIENCE

Teaching Assistant	**2020–2021**
San Diego State University	*San Diego, California*
Teaching Assistant	**2016–2017**
Illinois Institute of Technology	*Chicago, Illinois*
Teaching Assistant	**2013–2014**
University of Wisconsin, La Crosse	*La Crosse, Wisconsin*

REFEREED JOURNAL PUBLICATIONS

Process scenario discovery from event logs based on activity and timing information **2022**
Journal of Systems Architecture

UACFinder Mining Syntactic Carriers of Unspecified Assumptions in Medical Cyber-Physical System Design Models **2020**
ACM Transactions on Cyber-Physical Systems

A framework for supporting the development of verifiably safe medical best practice guideline systems **2020**
Journal of Systems Architecture

REFEREED CONFERENCE PUBLICATIONS

Using Domain Knowledge to Assist Process Scenario Discoveries **Jun 2022**
2022 IEEE 46th Annual Computers, Software, and Applications Conference

Empirical Studies of Three Commonly Used Process Mining Algorithms **Oct 2021**
2021 IEEE International Conference on Systems, Man, and Cybernetics

Improving Process Discovery Results by Filtering Out Outliers from Event Logs with Hidden Markov Models **Sep 2021**
2021 IEEE 23rd Conference on Business Informatics

Using Event Log Timing Information to Assist Process Scenario Discoveries **Dec 2020**
2020 IEEE Third International Conference on Artificial Intelligence and Knowledge Engineering

Mining timing constraints from event logs for process model **Jun 2020**
2020 IEEE 44th Annual Computers, Software, and Applications Conference

Prevent Potential Hazards Caused by Medical Device Time Differences in Integrated Clinical Environments **Jun 2019**
2019 IEEE International Conference on Embedded Software and Systems

Reducing Patient Waiting Time for Radiotherapy Treatments with a Genetic Algorithm **Dec 2018**
2018 International Conference on Computational Science and Computational Intelligence

IAfinder: Identifying potential implicit assumptions to facilitate validation in medical cyber-physical system **Jun 2018**
2018 55th ACM/ESDA/IEEE Design Automation Conference

Model and integrate medical resource available times and relationships in verifiably correct executable medical best practice guideline models **Apr 2018**
2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems

SOFTWARE

Timing Constraint Discovery tool
`https://github.com/stevenzzy9/timingConstraintsMining.`
Java algorithm that discovers timing constraints from event logs.

Process Scenarios Discovery tool
`https://github.com/stevenzzy9/WastewaterTreatmentProcessDiscovery.`
Java algorithm that discovers process scenarios related to the wastewater treatment process from event logs.

ABSTRACT OF THE DISSERTATION

A Framework for Discovering Implicit Knowledge from Event Logs

By

Zhenyu Zhang

Doctor of Philosophy in Computational Science

University of California, Irvine, 2022

Professor Shangping Ren, Chair

Although automation has become widespread in many industries, some workplaces, such as utility industries, still rely heavily on individuals to perform critical tasks based on their extensive past experiences. This accumulated intellectual capital not only improves productivity and efficacy under normal conditions but more importantly, also contains knowledge about identifying anomalies and addressing unexpected events and situations. This knowledge is essential and critical to ensuring system safety and reliability, especially for industries with aging infrastructures where anomalies are becoming more common. Unfortunately, this knowledge is not explicitly recorded in defined guidelines, protocols, or standard workflows but implicitly resides in the minds of skilled workers and routine event logs. As skilled workers retire or leave the business, we may lose this accrued indispensable knowledge that is critical to the industries' productivity, reliability, and safety.

This thesis proposes a framework to discover implicit knowledge from event logs. The implementation of the framework contains three phases. We first proposed an approach that uses hidden Markov models to filter out outliers from event logs in the clean phase. Then, in the discovery phase, we propose approaches to discover the implicit knowledge related to timing constraints and process scenarios from event logs. For timing constraint discovery, we have presented an approach that extends existing process mining techniques to mine and integrate timing constraints with a workflow or process model constructed by any existing process mining algorithm. A real-life road traffic fine management process scenario is used as a case study to investigate the effectiveness and validity of the approach. For process scenario discovery, we present the distance-based and density-based approaches that obtain timing information from event logs and use the information to assist process scenario discoveries. A wastewater treatment process provided by a domain expert is used as a case study to investigate the effectiveness and validity of the approach. In the incorporating phase, we present an approach that incorporates domain knowledge to assist in discovering process scenarios from event logs. The case studies and experiment results indicate that the proposed implicit knowledge discovery framework can mine implicit knowledge related to the timing constraints and process scenarios from event logs and outperform commonly used approaches with different real-life event logs.

Chapter 1

Introduction

1.1 Background

As with many developed nations, the United States is amidst an aging workforce crisis. According to labor force projections from the U.S. Bureau of Labor Statistics, workers aged 55 and older will not only be the fastest growing segment of the labor market in the coming decade [66], but will also make up 25.2% of the total workforce by 2024 compared to 13.1% in 2000 [67]. As these workers near or surpass typical retirement ages, concerns are centered on how to prepare the workforce for the loss of decades of accumulated knowledge and skills. Such concerns are exacerbated by the exceptional size of the workforce nearing retirement over the next few decades; by 2024, approximately 41 million workers will be over the age of 55, among which approximately 42% of them are in management and professional occupations [67].

Although automation has become widespread in many industries, some workplaces, such as utility industries, still rely heavily on individuals to perform critical tasks based on their extensive past experiences. This accumulated intellectual capital not only improves productivity and efficacy under normal conditions, but more importantly, also contains knowledge about identifying anomalies and addressing unexpected events and situations. This knowledge is essential and critical to ensuring system safety and reliability, especially for industries with aging infrastructures where anomalies are becoming more common. Unfortunately, this knowledge is not explicitly recorded in defined guidelines, protocols, or standard workflows, but implicitly resides in the minds of skilled workers. As skilled workers retire or leave the business, we may lose this accrued indispensable knowledge that is critical to the industries' productivity, reliability, and safety. For instance, the Gerdau Group, the biggest producer of long steel operating in the Americas, encountered a situation where it risked significant knowledge loss [44]. An experienced and senior worker was suddenly leaving the business for health reasons, with no one who would be left within the company with the same knowledge and skills. This worker had an implicit knowledge about a procedure -whether to turn off the furnace during shutdowns in production. It was a long time before the other workers concluded that it was best to keep the furnace burning at a temperature of roughly $13300F$. The reason is that it would cost more fuel to reheat the furnace after it was turned off rather than keep it running at a lower temperature. However, before that, the workers frequently changed the furnace's temperature, damaging the refractory bricks and shortening the wall life of the furnaces by 25%. The loss of implicit knowledge resulted in a loss of $25,000 per furnace for the company. Hence, accurately capturing and retaining critical experiences and implicit knowledge has become an urgent problem.

Over the past two decades, many process control systems have been developed in many areas, including industrial manipulation and production, medicine, the military, and agriculture [9, 21, 42, 39]. Most existing industrial process control systems maintain event logs to understand what actually happened to a system [62]. Event logs record amounts of process execution data, such as time information, the actual event description, the event's severity, the executor or process involved, or other relevant information. As a large amount of event logs are generated, the implicit knowledge not only exists in the minds of skilled workers but is also hidden in event logs. Process mining is an emerging research field that looks at event logs to build graphical models and provides new insights to businesses that allow them to make process-driven decisions. Process mining aims to use event data to extract process-related information, e.g., to automatically discover a process model by observing events recorded by some enterprise system [68]. Hence, process mining is a possible solution for accurately capturing and retaining critical experiences and implicit knowledge.

The term ‘implicit knowledge’ was first used by chemical engineer turned scientist Michael Polanyi in his 1958 book Personal Knowledge: Towards a Post-Critical Philosophy [54]. In this book, implicit knowledge is defined as the knowledge gained through living and working experience, both in personal life and professional development. Implicit knowledge is abstract, subjective, informal, and difficult to share or express. Today, implicit knowledge is often expressed in behaviors, timing constraints, routines, practical process scenarios, responses, etc. Discovering and retaining all types of implicit knowledge is complex and timing-consuming. Hence, we choose timing constraints and practical process scenarios as the focus of this thesis. The reasons are as follows: (1) the timing information can be critical in many domains,and timing constraints need to be enforced in order to provide safe and effective manipulation; (2) over time, the process evolves into a complicated comprehensive system. There is a gap between the practical process, including multiple scenarios, and the defined process. Understanding the practical process scenarios is critical to the industries' productivity, reliability, and safety. Therefore, the thesis addresses the challenges of applying process mining techniques to discover implicit knowledge in terms of timing constraints and practical process scenarios from event logs.

1.2 Technical Challenges

Process discovery is the first step of process mining. Process discovery is focused on producing a process model from behavior seen and captured within an event log, which means it takes an input of an event log, applies a process mining algorithm, and then provides an output of a process model. The resulting model can help to obtain the knowledge hidden in event logs. The next step of process mining is conformance checking, which requires an event log and a process model as input. Its primary goal is to observe and determine discrepancies between the model passed in and what is actually observed in the event log [74]. Additionally, conformance checking is used to measure the performance of process discovery algorithms and uncover models that are not properly aligned with the actual behavior in a business environment. The last area of process mining is process enhancement, which focuses on extending and improving an existing process model using information about the actual processes record in an event log [74]. However, since most research efforts have been focused on improving discovery results, the existing works in process mining research are not adequate to directly uncover the implicit knowledge from event logs. The major technical challenges of applying process mining techniques to discover implicit knowledge from event logs are as follows.

Challenge I: Filtering Out Outliers from Event Logs

Many process mining algorithms are based on the assumption that event logs contain accurate representations of an ideal set of processes. These ideal sets of processes imply that the information contained within the log represents what is really happening in a given environment. However, many of these event logs might contain noisy, infrequent, missing, or false process information that are generally classified as outliers. The presence of outliers will lead to infrequent paths within the derived model. This causes the process model to become cluttered and results in a model that are simply not an accurate representation of the actual behavior. In order to limit these adverse effects, event logs are typically subjected to a pre-processing phase where they are manually cleaned from outliers [69]. However, this is a challenging and time-consuming task, with no guarantee on the effectiveness of the resulting model, especially in the context of large event logs exhibiting complex process behavior [63]. The inability to effectively detect and filter out outliers adversely affects the quality of the discovered model. In particular, the discovered model's precision can be greatly affected. Hence, we need to develop an approach to automatically and efficiently filter out outliers from event logs.

Challenge II: Mining Timing Constraints from Event Logs

Over the last decades, several research groups have been working on techniques for automated process discovery based on event logs. The goal of many process mining algorithms, such as Alpha algorithm [70], region-based approaches [60, 13], heuristic approach [80], to name a few, is to construct a process model from a set of event logs. However, these algorithms focus only on the functional aspects of a process, the timing information as to when an action must take place is neglected. The timing constraints can be one critical type of implicit knowledge in many domains. For instance, a silicon wafer undergoes a fabrication process by entering multiple production steps, where each step is performed by different, highly sophisticated tools. The skilled workers follow timing constraints at the wet etch and furnace process steps in order to prevent the likelihood of oxidation and contamination. Failing to do so, risks contact failures, low and unstable yields, the consequence of which is either rework or the wafers must be scrapped. Discovering timing constraints and following them has a considerable impact not only on a fab production performance but also its profitability. However, process models uncovered by most existing process mining techniques only reveal the underline structures of an actual process, and the timing constraints are not reflected in the process models. Therefore, we need to provide an effective way to mine the timing constraints from event logs.

Challenge III: Discovering Process Scenarios from Event Logs

Like the human body, an industrial control system can be broken down into a series of process scenarios, each performed by functionally related components working together to form a complex whole. A layperson may walk onto a plant floor and see complete systems at work, but the skilled worker takes a step closer to see the different process scenarios that make up the big picture, evaluates how each fits into the process, and addresses the unique requirements of the application. This skilled worker's ability to accurately distinguish between different process scenarios is essential in the industry.

Despite the demonstrated usefulness of process discovery algorithms, they face challenges in an environment where different scenarios exist [32, 15, 77, 35]. When different scenarios are grouped into one process model, the accuracy of the model representing reality reduces, but, more importantly, the complexity of the model becomes incomprehensible. This results in it being difficult, if not impossible, to achieve the goal of better understanding, monitoring and improving the current processes. Hence, we need to discover an approach to assist in discovering process scenarios from event logs.

Challenge IV: Using Domain Knowledge to Assist Implicit Knowledge Discoveries

While process scenarios discovery algorithms have proven to be valuable in multiple contexts, the techniques' largest drawback is most algorithms achieve this by solely using an event log without allowing the domain expert to influence the discovery in any way. However, the domain expert has particular domain expertise that should be exploited to create better process models in model fitness and precision criteria. Understanding our data within the context of the problem we are trying to solve is crucial before we move on to discovering. Domain knowledge can help us understand how our data are collected and hence, the appropriate methods for preprocessing. With domain knowledge, we will also have guidance on what mainstream behaviors, representing the primary process scenario, might be helpful for implicit knowledge discoveries. Suppose we try to uncover the practical process scenarios from event logs related to wastewater treatment processes. A domain expert might articulate that when discovering the process scenarios in the wasterwater treatment process, we need to correctly identify the mainstream behaviors related to the liquid treatment process by microfiltration membrane bioreactors. Based on the domain knowledge, we could get better process models in model fitness and precision criteria. Hence, we need to incorporate domain knowledge to assist in discovering implicit knowledge from event logs.

This thesis proposes a framework to address these challenges in discovering implicit knowledge from event logs. The following section provides a high-level overview of the framework's design and explains how these challenges are addressed.

1.3 Framework Architecture

In order to address the four technical challenges presented in Chapter 1.2, the thesis designs a framework to discover implicit knowledge from event logs. The architecture of the framework is depicted in the following Figure 1.1. The implementation of the framework contains three phases. A brief introduction of each phase is as follows.

Cleaning Phase: This phase takes the raw event logs as the input, and the clean event logs are the output. It contains an automatic process that uses hidden Markov models to filter out outliers from event logs prior to applying process discovery algorithms to improve process discovery results. The approach has been implemented on top of the *pm4py* Framework [7] We then analyze if filtering out outliers by the proposed approach can improve the quality of the implicit knowledge in the form of process models. Through the analysis, we find that our approach adequately identifies and removes outliers, leading to an increase in the process

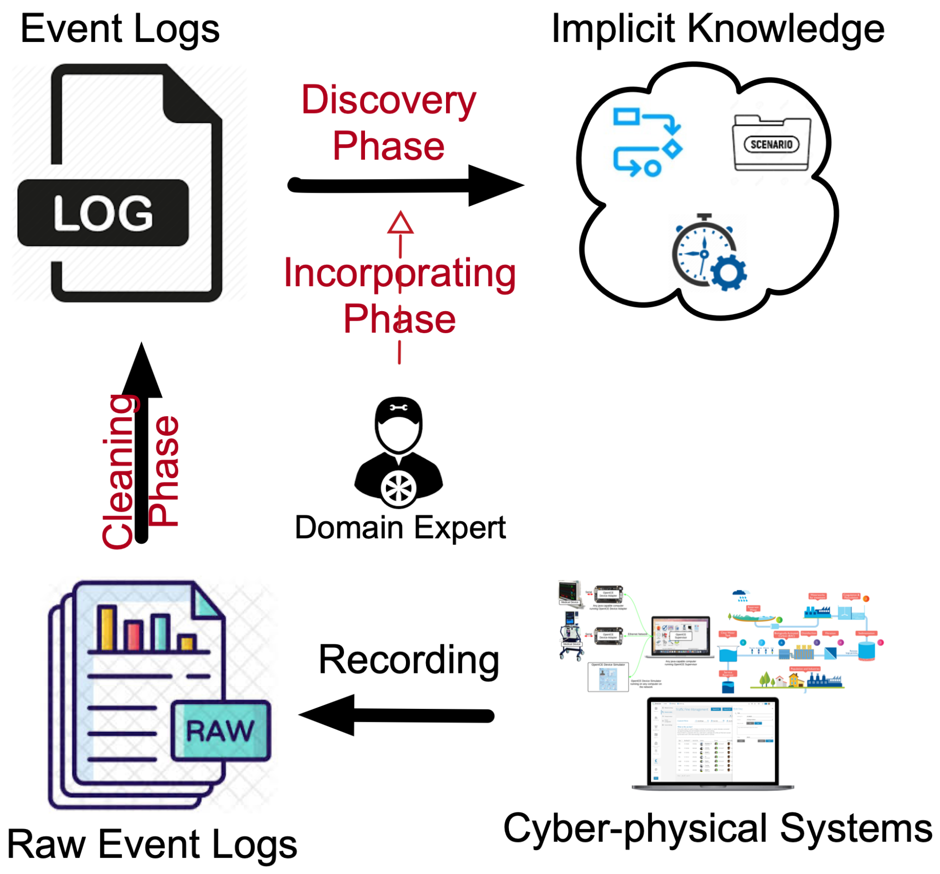


Figure 1.1: The architecture of the framework for discovering implicit knowledge from event logs

discovery results. We also evaluate the proposed approach by using the combination of two commonly used filtering approaches: the *Matrix Filter* approach [58] and the *Anomaly Free Automation* approach [20]. The evaluation results show that the proposed method outperforms two commonly used filtering approaches for both artificial and real-life event logs.

Discovery Phase: For this phase, we propose approaches to discover the implicit knowledge related to timing constraints and process scenarios from event logs. This phase takes the clean event logs as the input. The implicit knowledge related to timing constraints or process scenarios is output represented by process models.

For timing constraints discovery, we have presented an approach that extends existing process mining techniques to mine and integrate timing constraints with process models. A real-life road traffic fine management process scenario is used as a case study to investigate the effectiveness and validity of the approach. The evaluation results show that the algorithm is able to discover the timing constraints of an actual workflow process from event logs.

For process scenarios discovery, we propose a distance-based approach and a density-based approach, both of which obtain temporal information from event logs and use this information to assist process scenario discoveries. A real wastewater treatment process provided by a domain expert is used as a case study to investigate the effectiveness and validity of these two approaches. We also use real-life event logs to compare the performance of the proposed approaches for process scenario discoveries with the commonly used k-means clustering approach. The experiment data shows that these two proposed approaches are able to discover the process scenarios from event logs.

Incorporating Phase: In the incorporating phase, we present an approach incorporating domain knowledge to discover implicit knowledge related to process scenarios from event logs. The algorithm is implemented based on the open-source process mining framework pm4py. We examine the effectiveness of the proposed approach for process scenario discovery in the wastewater treatment domain. And then, we evaluate whether incorporating domain knowledge can improve the process scenario discovery performance. The experiment data show that the proposed approach is able to discover the process scenarios from event logs by incorporating domain knowledge, and the process models obtained with the proposed approach have higher performance.

In summary, the successful implementation addresses all the challenges identified above. More specifically, Challenge I is addressed by the *Cleaning Phase*. Challenge II and Challenge III are addressed by the *Discovery Phase*. Challenge IV is addressed by the *Incorporating Phase*.

1.4 Thesis Outline

Chapter 3 presents an approach that uses Hidden Markov Models to filter out outliers from event logs before applying any process discovery algorithms. In Chapter 4, we present an approach that extends existing process mining techniques to mine and integrate timing constraints with a workflow or process model constructed by any existing process mining algorithm. In Chapter 5, we discover distance-based and density-based approaches that use timing information to assist in discovering process scenarios from event logs. In Chapter 6, we propose an approach that incorporates domain knowledge to assist in discovering process scenarios from event logs. Every technical chapter, i.e., Chapters 3-6, contains sections to discuss the background and related work and illustrate the experiments, respectively. We conclude the thesis in Chapter 7 with a summary of the work and open challenges.

Chapter 2

Definitions and Notations

Discovering implicit knowledge is impossible without proper data. Hence, we formalize notations and definitions related to event logs used in this thesis. Then, preliminaries are presented, including various process modeling notations and process discovery techniques.

2.1 Event Logs

The starting point of implicit knowledge discovery is event logs which record information about activities as they take place. An event log is a file containing recorded business or process information that gets captured over a period of time. The standard file type for an event log is an XES file. This file type has been approved as the standard by IEEE, but many event logs are also composed of comma-separated value files. In this part, we will use a simple event log provided by the open-source process mining framework for illustrative purposes. In Figure 2.1, we see the XES formatted event log, which distinguishes between events and traces via data identifiers, i.e., $< event >$ and $< \backslash event >$. Notice that there is metadata at the top of the log before the first instance of a trace. This data describes the attribute mappings that are observed in the event and trace data. Additionally, this example does not have any case-level attributes, only event-level. To make this file more understandable, we convert the values into a table view as shown in Table 2.1. Each row in this table represents an event linked to a trace via the case id attribute in the first column.

For the formal definitions, we adopt definitions similar to the ones given in [68]. In particular, for a given activity set $\mathit{\Omega}$, an event entry e in an event log records an activity happening within the operation of a process. An event trace σ is a finite sequence of event entries ordered by their occurrence time. An event log log consists of a set of event traces. The formal definitions of an event entry, an event trace, and an event log are given below.

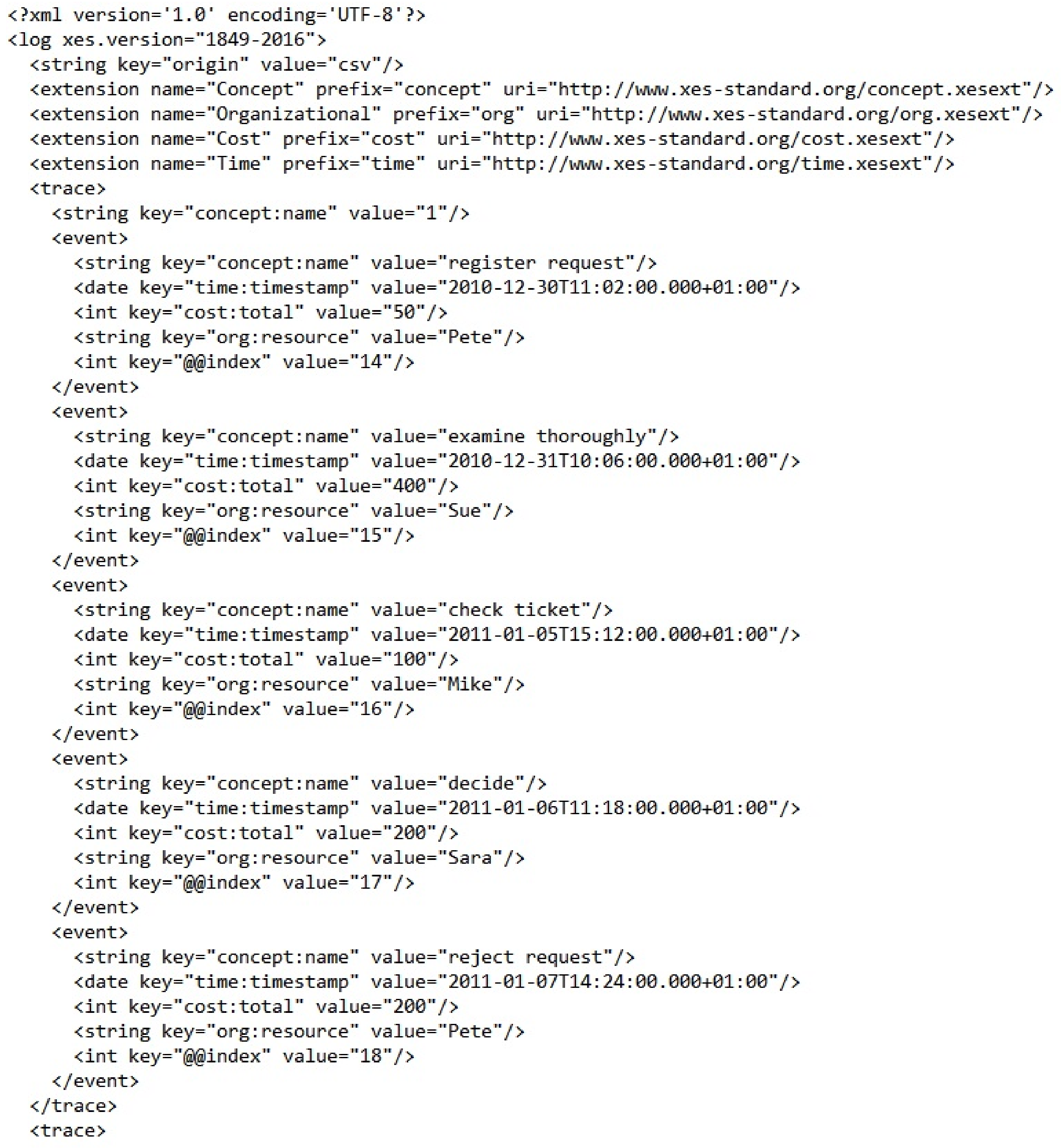
Definition 1 (Event Entry (e))**.** *An event entry e is a tuple (α, τ), i.e., $e = (\alpha, \tau)$, where α is activity's name and τ is the timestamp of activity α. The sets of all event activity names, activity timestamps, and events in a given event log are denoted as $\mathit{\Omega}$, $\mathcal{T}$ and $\mathcal{E}$, respectively.*

Definition 2 (Attribute Function ($\mathcal{F}_\alpha$ and $\mathcal{F}_\tau$))**.** *The attribution functions are defined to obtain an event entry's activity name and time stamps, respectively, i.e., $\mathcal{F}_\alpha : \mathcal{E} \mapsto \mathit{\Omega}$ and $\mathcal{F}_\tau : \mathcal{E} \mapsto \mathcal{T}$.*

Definition 3 (Event Trace (σ))**.** *An event trace σ is a finite sequence of event entries $e_1, \cdots, e_n$, i.e., $\sigma = [e_1, \cdots, e_n]$, where $\mathcal{F}_\tau(e_i) < \mathcal{F}_\tau(e_{i+1})$ and $1 \leq i \leq n$.*

Definition 4 (Event Log (log))**.** *An event log* log *is a set of event traces, i.e.,* $\mathrm{log} = \{\sigma_1, \cdots, \sigma_m\}$. *The number of traces in the event log* log *is denoted as* $|\log|$.

For the Table 2.1. This event log log has five traces, i.e., $\log = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\}$ and $|\log| = 6$. The log also shows that for trace σ_1, there are five event entries, i.e., $\sigma_1 = [e_1, e_2, e_3, e_4, e_5]$. It is worth pointing out that the ordering of event entries within a trace is important, while the ordering of event entries among different traces is of no significance. The activity name and its timestamp of an event entry in an event trace are obtained by



```
<?xml version='1.0' encoding='UTF-8'?>
<log xes.version="1849-2016">
  <string key="origin" value="csv"/>
  <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
  <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
  <extension name="Cost" prefix="cost" uri="http://www.xes-standard.org/cost.xesext"/>
  <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
  <trace>
    <string key="concept:name" value="1"/>
    <event>
      <string key="concept:name" value="register request"/>
      <date key="time:timestamp" value="2010-12-30T11:02:00.000+01:00"/>
      <int key="cost:total" value="50"/>
      <string key="org:resource" value="Pete"/>
      <int key="@@index" value="14"/>
    </event>
    <event>
      <string key="concept:name" value="examine thoroughly"/>
      <date key="time:timestamp" value="2010-12-31T10:06:00.000+01:00"/>
      <int key="cost:total" value="400"/>
      <string key="org:resource" value="Sue"/>
      <int key="@@index" value="15"/>
    </event>
    <event>
      <string key="concept:name" value="check ticket"/>
      <date key="time:timestamp" value="2011-01-05T15:12:00.000+01:00"/>
      <int key="cost:total" value="100"/>
      <string key="org:resource" value="Mike"/>
      <int key="@@index" value="16"/>
    </event>
    <event>
      <string key="concept:name" value="decide"/>
      <date key="time:timestamp" value="2011-01-06T11:18:00.000+01:00"/>
      <int key="cost:total" value="200"/>
      <string key="org:resource" value="Sara"/>
      <int key="@@index" value="17"/>
    </event>
    <event>
      <string key="concept:name" value="reject request"/>
      <date key="time:timestamp" value="2011-01-07T14:24:00.000+01:00"/>
      <int key="cost:total" value="200"/>
      <string key="org:resource" value="Pete"/>
      <int key="@@index" value="18"/>
    </event>
  </trace>
  <trace>
```

Figure 2.1: View of the XES event log file which displays the first trace observed in the log

function $\mathcal{F}_\alpha$ and $\mathcal{F}_\tau$, respectively. For instance, the activity name of e_2 in trace σ_1 is $\mathcal{F}_\alpha(e_2) = examinethoroughly$, and its timestamp is $\mathcal{F}_\tau(e_2) =$ "$2010 - 12 - 31\ 10 : 06 : 00 + 01 : 00$".

2.2 Process Models

The plethora of process modeling notations available today illustrates the importance of process modes. Some industries may use only informal process models to structure discussions and document process-related knowledge. As business processes have become more complex, heavily rely on process control systems, and may span multiple organizations, process modeling has become of the utmost importance. Process models assist in managing complexity by providing insight and documenting knowledge related to procedures, timing constraints, resource management, etc. Today, most process models are made by hand and are not based on rigorous analysis of existing process data. Hence, this thesis focuses on the process model that can be generated by process mining techniques. This section presents preliminaries that will be used in later chapters. In particular, various process modeling definitions and notations are introduced, and some analysis techniques are reviewed.

Petri nets are the oldest and best investigated process modeling language allowing for the modeling of concurrency. Although the graphical notation is intuitive and simple, Petri nets are executable and many analysis techniques can be used to analyze them [41, 26, 87]. A Petri net is a bipartite graph consisting of places and transitions. The network structure is static, but, governed by the firing rule, tokens can flow through the network. The state of a Petri net is determined by the distribution of tokens over places and is referred to as its marking. Hence, we choose the Petri net as representation for implicit knowledge related to the workflows and process scenarios.

Definition 5 (Petri net [26]). *A Petri net N is a tuple (P, T, F), where*

case id	activity	timestamp	costs	resource
3	register request	2010-12-30 14:32:00+01:00	50	Pete
3	examine casually	2010-12-30 15:06:00+01:00	400	Mike
3	check ticket	2010-12-30 16:34:00+01:00	100	Ellen
3	decide	2011-01-06 09:18:00+01:00	200	Sara
3	reinitiate request	2011-01-06 12:18:00+01:00	200	Sara
3	examine thoroughly	2011-01-06 13:06:00+01:00	400	Sean
3	check ticket	2011-01-08 11:43:00+01:00	100	Pete
3	decide	2011-01-09 09:55:00+01:00	200	Sara
3	pay compensation	2011-01-15 10:45:00+01:00	200	Ellen
2	register request	2010-12-30 11:32:00+01:00	50	Mike
2	check ticket	2010-12-30 12:12:00+01:00	100	Mike
2	examine casually	2010-12-30 14:16:00+01:00	400	Sean
2	decide	2011-01-05 11:22:00+01:00	200	Sara
2	pay compensation	2011-01-08 12:05:00+01:00	200	Ellen
1	register request	2010-12-30 11:02:00+01:00	50	Pete
1	examine thoroughly	2010-12-31 10:06:00+01:00	400	Sue
1	check ticket	2011-01-05 15:12:00+01:00	100	Mike
1	decide	2011-01-06 11:18:00+01:00	200	Sara
1	reject request	2011-01-07 14:24:00+01:00	200	Pete
6	register request	2011-01-06 15:02:00+01:00	50	Mike
6	examine casually	2011-01-06 16:06:00+01:00	400	Ellen
6	check ticket	2011-01-07 16:22:00+01:00	100	Mike
6	decide	2011-01-07 16:52:00+01:00	200	Sara
6	pay compensation	2011-01-16 11:47:00+01:00	200	Mike
5	register request	2011-01-06 09:02:00+01:00	50	Ellen
5	examine casually	2011-01-07 10:16:00+01:00	400	Mike
5	check ticket	2011-01-08 11:22:00+01:00	100	Pete
5	decide	2011-01-10 13:28:00+01:00	200	Sara
5	reinitiate request	2011-01-11 16:18:00+01:00	200	Sara
5	check ticket	2011-01-14 14:33:00+01:00	100	Ellen
5	examine casually	2011-01-16 15:50:00+01:00	400	Mike
5	decide	2011-01-19 11:18:00+01:00	200	Sara
5	reinitiate request	2011-01-20 12:48:00+01:00	200	Sara
5	examine casually	2011-01-21 09:06:00+01:00	400	Sue
5	check ticket	2011-01-21 11:34:00+01:00	100	Pete
5	decide	2011-01-23 13:12:00+01:00	200	Sara
5	reject request	2011-01-24 14:56:00+01:00	200	Mike
4	register request	2011-01-06 15:02:00+01:00	50	Pete
4	check ticket	2011-01-07 12:06:00+01:00	100	Mike
4	examine thoroughly	2011-01-08 14:43:00+01:00	400	Sean
4	decide	2011-01-09 12:02:00+01:00	200	Sara
4	reject request	2011-01-12 15:44:00+01:00	200	Ellen

Table 2.1: A visual representation of the pm4py example event log

- *P is a finite set of places;*
- *T is a finite set of transitions, $P \cap T = \emptyset$; and*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs.*

Notation 1 ($\bullet t$)**.** *Given a Petri net $N = (P, T, F)$ and a transition $t \in T$, we use notation $\bullet t$ to represent a set of places immediately before transition t, i.e., $\bullet t = \{p | p \in P \wedge (p, t) \in F\}$.*

Notation 2 ($t\bullet$)**.** *Given a Petri net $N = (P, T, F)$ and a transition $t \in T$, we use notation $t\bullet$ to represent a set of places immediately after transition t, i.e., $t\bullet = \{p | p \in P \wedge (t, p) \in F\}$.*

The state of a Petri net $N = (P, T, F)$ is represented by its markings which is a distribution of tokens on places P. A marking of N is defined by a mapping function $m : P \rightarrow \mathbb{N}$, where $\mathbb{N}$ is the natural number set. A place p is *marked* by a marking m if $m(p) > 0$.

The execution semantics of a Petri net $N = (P, T, F)$ are defined by transition firings which specify the enabling conditions and the marking transformation of the Petri net. A transition $t \in T$ is enabled by a marking m if m marks all places in its pre-set $\bullet t$, i.e.,

$$\forall p \in \bullet t : m(p) > 0. \tag{2.1}$$

The firing of an enabled transition t transforms the marking m to m' as below:

$$m'(p) = \begin{cases} m(p) - 1 & \texttt{if} \quad p \in \bullet t \wedge p \notin t\bullet \\ m(p) + 1 & \texttt{if} \quad p \notin \bullet t \wedge p \in t\bullet \\ m(p) & \texttt{otherwise} \end{cases} \tag{2.2}$$

We use the Petri net shown in Figure 2.2 to explain the above concepts. In the graphical representation, *places*, *transitions*, *arcs*, and *tokens* are represented by circles, squares, arrows, and black dots, respectively. The Petri net N shown in Figure 2.2 is defined as (P, T, F), where $P = \{p_1, p_2, p_3, p_4\}$, $T = \{t_1, t_2\}$, and $F = \{(t_1, p_1), (p_1, t_2), (t_2, p_2),$ $(t_2, p_3), (t_2, p_4), (p_2, t_1)\}$. The current marking is $m = \{(p_1, 3), (p_2, 0), (p_3, 0), (p_1, 1)\}$. Based on formula (2.1), the transition t_2 is enabled by the marking m. After the firing of t_2, according to formula (2.2), the new marking becomes $m' = \{(p_1, 2), (p_2, 1), (p_3, 1), (p_1, 2)\}$.

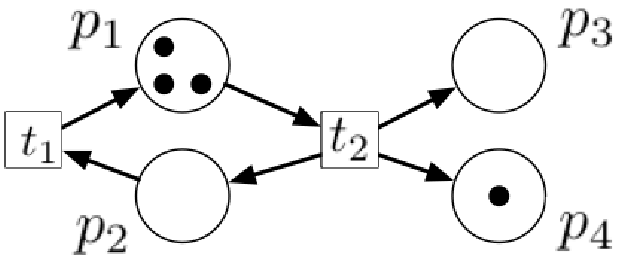


Figure 2.2: Simple Petri net example with two transitions and four places

A good reason for using Petri nets to model workflows is that Petri nets treat states and events on an equal footing [72]. They are event-based and state-based at the same time. Their formal analysis methods allow extraction and calculations on either of these aspects in isolation or in combination where appropriate. This results in a subclass of the Petri net named a *workflow* net with three further constraints: (1) there is one and only one *input place* where a process starts; and (2) there is one and only one *output place* where the process ends; and (3) all elements are on a path from the input place to the output place. The formal definition is given below.

Definition 6 (*Workflow net* [73]). *A net $N = (P, T, F)$ is a workflow net, if it is a Petri net and satisfies the following constraints:*

- *There is one and only one input place i, i.e.*

  1. $\exists i \in P$, *s.t.* $\forall t \in T, (t, i) \notin F$,

  2. $\exists i_1, i_2 \in P$, $(\forall t \in T, (t, i_1) \notin F \land (t, i_2) \notin F) \to i_1 = i_2$.

- *There is one and only one output place o, i.e.*

  1. $\exists o \in P$, *s.t.* $\forall t \in T, (o, t) \notin F$,

2. $\exists o_1, o_2 \in P$*,* $(\forall t \in T, (o_1, t) \notin F \wedge (o_2, t) \notin F) \rightarrow o_1 = o_2$

- *For a pseudo transition* $\xi \notin T$*, the net* $(P, T \cup \{\xi\}, F \cup \{((o, \xi), (\xi, i)\})$ *is a strongly connected net.*

Timing constraints in process control system is crucial in determining and controlling the life cycle of activities. Most integration of timing constraints with process models involve the assignment of deadlines and other external temporal and synchronisation constraints, the calculation of the overall process duration and the checking of timing inconsistencies [27, 50]. However, in our work, we want to determine when the task is enabled, when it is executed and for how long the execution will last, not a timing constraint on the overall process. Hence, we choose time Petri netas representation for implicit knowledge related to the timing constraints [47]. The time Petri nethas dealt with the effect of temporal constraints at every activity and how these constraints affect the liveness and safety aspects of the entire process. More specifically, the time Petri netextends Petri net by specifying a time interval for every transition to constrain firing duration. The definition is as follows.

Definition 7 (Time Petri Net [47])**.** *A time Petri net* N *is a tuple* (P, T, F, I) *such that* (P, T, F) *is a Petri net and* I *associates each transition* $t \in T$ *with a static firing time interval, i.e.,* $I : T \rightarrow \{[a, b] \in \mathbb{R}^* \times (\mathbb{R}^* \cup +\infty) | a \leq b\}$*, where* $\mathbb{R}^*$ *is the set of all non-negative real numbers.*

Given a time Petri net $N = (P, T, F, I)$, the firing time interval $I(t) = [a, b]$ for transition $t \in T$ specifies the earliest firing time a and the latest firing time b after the transition t is enabled. For instance, suppose the transition t is enabled at time instance τ, t can only be fired during time interval $[\tau + a, \tau + b]$. If the transition t is not fired during $[\tau + a, \tau + b]$, then t becomes disabled.

If a transition t's firing time interval is $I(t) = [0, +\infty]$, it indicates that t can be fired immediately after enabling and never becomes disabled if the N's state does not change.

Hence, if $\forall t \in T : I(t) = [0, +\infty]$, the given time Petri net $N = (P, T, F, I)$ has equivalent execution semantics with Petri net $N' = (P, T, F)$.

Recently, the Business Process Modeling Notation (BPMN) has become one of the most widely used languages to model processes [84]. The BPMN is supported by many tool vendors used in many industries and has been standardized by the OMG [51]. Most existing standard guidelines in utility industries are represented by BPMN models, such as wastewater treatment process model [52], air traffic process control systems [86]. Since the BPMN is to support process management by providing a notation that is intuitive to users, domain experts from different disciplines can easily understand and produce the BPMN model. Hence, in this thesis,we choose the BPMN model as the representation of the domain knowledge.

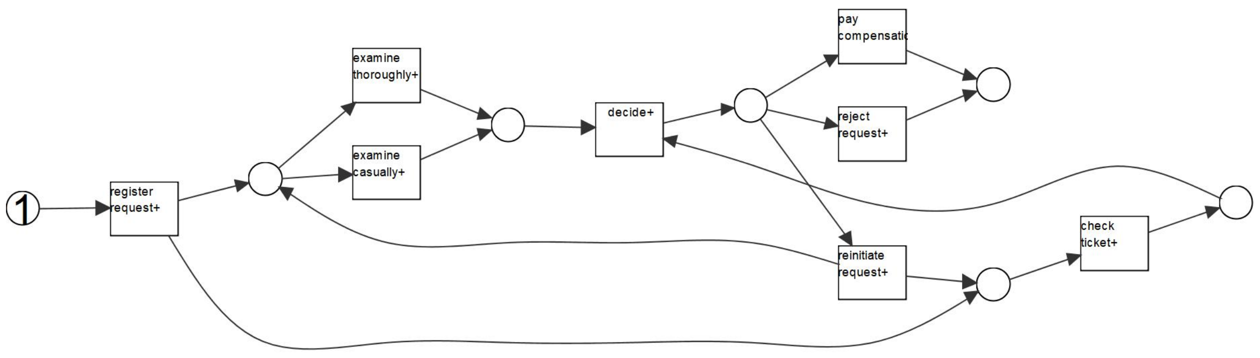


Figure 2.3: Process model using the **Alpha Algorithm** from data in Table 2.1.

2.3 Process Discovery Algorithms

Process Discovery describes the data-based visualization of a process using process mining discovery algorithms. In the last two decades, several process discovery algorithms have been

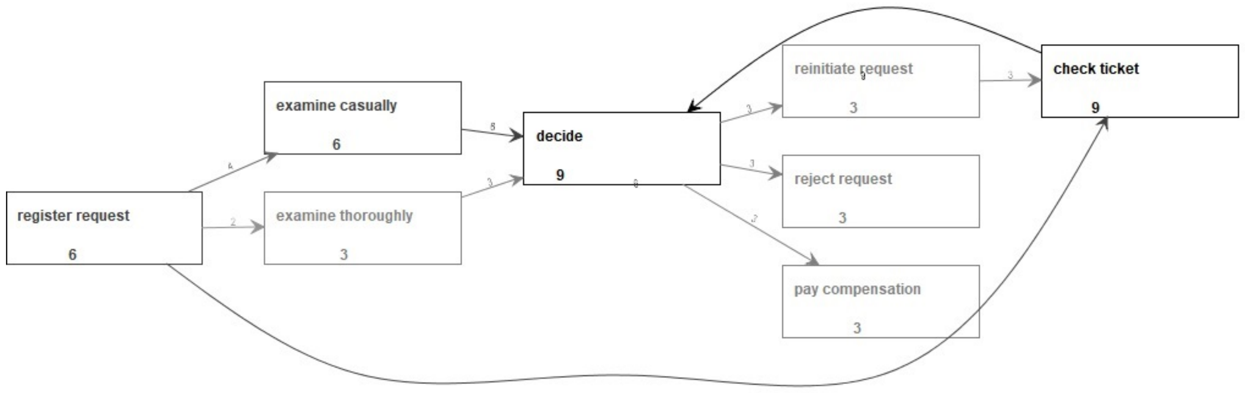


Figure 2.4: Process model using the **Heuristic Algorithm** from data in Table 2.1.

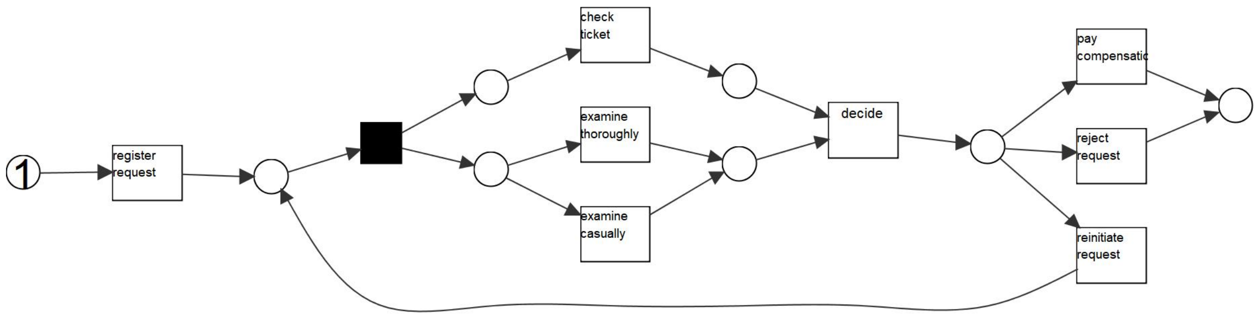


Figure 2.5: Process model using the **Inductive Algorithm** from data in Table 2.1.

proposed [23, 24, 36, 46, 70, 80, 82, 83]. These algorithms are based upon or are extensions of three commonly used process mining algorithms: the α algorithm [70], the Heuristic algorithm [80], and the Inductive algorithm [46]. The input of these process discovery algorithms is an event log, and the output is a process model. We will briefly introduce these three process discovery algorithms.

The Alpha algorithm was one of the first process discovery algorithms [71]. The algorithm first scans the event log for particular patterns. These patterns are called log-based ordering relationships and are used to discover dependency patterns between events in the log. These patterns are represented in a footprint matrix which describes the constraints and relations each event has with respect to other observed events. The Figure 2.3 shows the results of the Alpha algorithm using the log from Table 2.1. The resulting Petri net is simple to understand and requires much less effort to analyze compared to the table view. However, the algorithm was not a particularly good tool for mining process scenarios because it was too sensitive to noise and outliers.

The Heuristic algorithm [80] is a Frequency-based approach that takes frequencies of events and sequences into account and constructs process models similar to causal nets. The Heuristic miner intends to focus on showing the mainstream behavior, which allows it to be more robust to noise. Additionally, this algorithm has a few key advantages over the Alpha miner. It takes frequency into account, which allows it to filter out exceptional or noisy behavior, it's able to detect short loops, and it allows a miner to skip single event activities. Using the same event log from Table 2.1, we construct a process model discovered by the Heuristic algorithm. The resulting process model is shown in Figure 2.4.

In recent years, the Inductive Miner and its variants have dominated process discovery applications. The Inductive Miner allows for many alternatives and variants, is excellent at handling infrequent behavior, and can deal with most large models. The most significant benefit of the inductive miner is that it results in a process tree that ensures formal correctness and avoids deadlocks, allowing for a model that is sound by construction [74, 45]. According to the divide and conquer strategy, the Inductive Miner works by repeatedly finding a split within a trace in the event log, detecting the operator that describes the split, and then further examining the resulting sublogs. This recursive nature continues until we have formed sublogs corresponding to the execution of a single activity [74]. To determine the order of splitting, the Inductive miner outlines rules for making "cuts” based on exclusive-choice cuts, parallel cuts, redo-loop cuts, and process tree operators. Figure 2.5 shows the results of applying the Inductive Miner [45] on the event log shown in Table 2.1.

Chapter 3

Event Logs Cleaning

3.1 Background and Related Work

Most process mining algorithms mentioned in Chapter 2.3 are based on the assumption that an event log accurately represents information about a working process as it takes place. Unfortunately, many real-life process event logs often contain *noise* and *infrequent behavior*. *Noise* refers to inserting erroneous activities in the log, not logging some activities that have occurred, or reporting some activities with an out-of-order time sequence [64, 65].

The presence of noise can result from data entry problems, faulty data collection instruments, data transmission, streaming problems, or other technical limitations. However, *infrequent behavior* refers to a behavior that only occurs in exceptional case within the process. Without having business or domain knowledge, distinguishing between noise and infrequent behavior is a challenging task [59]. Therefore, we consider this as a separate research question and do not cover such cases in this thesis. As such, we consider both noise and infrequent behavior to be *outliers*.

The presence of outliers will lead to infrequent paths within the derived model. This causes the process model to become cluttered and results in a model that is simply not an accurate representation of the actual behavior. In order to limit these adverse effects, event logs are typically subjected to a pre-processing phase where they are manually cleaned from outliers [69]. However, this is a challenging and time-consuming task, with no guarantee on the effectiveness of the resulting model, especially in the context of large event logs exhibiting complex process behavior [63].

The inability to effectively detect and filter out outliers adversely affects the quality of the discovered model. In particular, the discovered model's precision can be greatly affected. A frequently used performance metric is precision which shows the degree to which a model allows for unobserved behavior in the log. In some cases, low levels of outliers will have a detrimental effect on the quality of the models produced by various discovery algorithms. The Heuristics Miner [80], Fodina [76], and the Inductive Miner [14] algorithms claim to have noise-tolerant capabilities. However, they result in low-quality models in the presence of low-levels of outliers. For instance, the Heuristics Miner can have a 49% drop in precision when the amount of outliers corresponds to just 2% of the total log size [20].

To address these concerns, researchers within the process mining community have proposed some approaches. The ProM framework offers several plugins for filtering anomalous data. These filtering methods are based on activity frequencies/positions, specific event attributes, and prefix-based rules, i.e., whether a trace is a prefix of another trace in the given event log, etc. However, these plugins require user input and domain knowledge. As a result, several studies have leveraged various methods to remove the so-called outliers from event logs before implementing process discovery algorithms. Some of these works require a reference model to replay process instances and filter out outliers. However, these methods are not often applicable due to the unavailability of reference models.

On the other hand, several studies perform filtering based on sequence mining algorithms.

Some of these generic methods are based on creating data models to represent normal behavior. The constructed models are then used for filtering out anomalous traces. An approach of this type is proposed in [20], which creates an abstraction of the observed behaviors based on event transitions using an Anomaly-Free Automaton (AFA). Subsequently, this algorithm removes the infrequent event transitions from the constructed automaton, and considers non-replayable log traces on the so-called automaton as outlier traces. Although showing improvement in performance measures, this method cannot take incomplete traces or the ones with missing events into account.

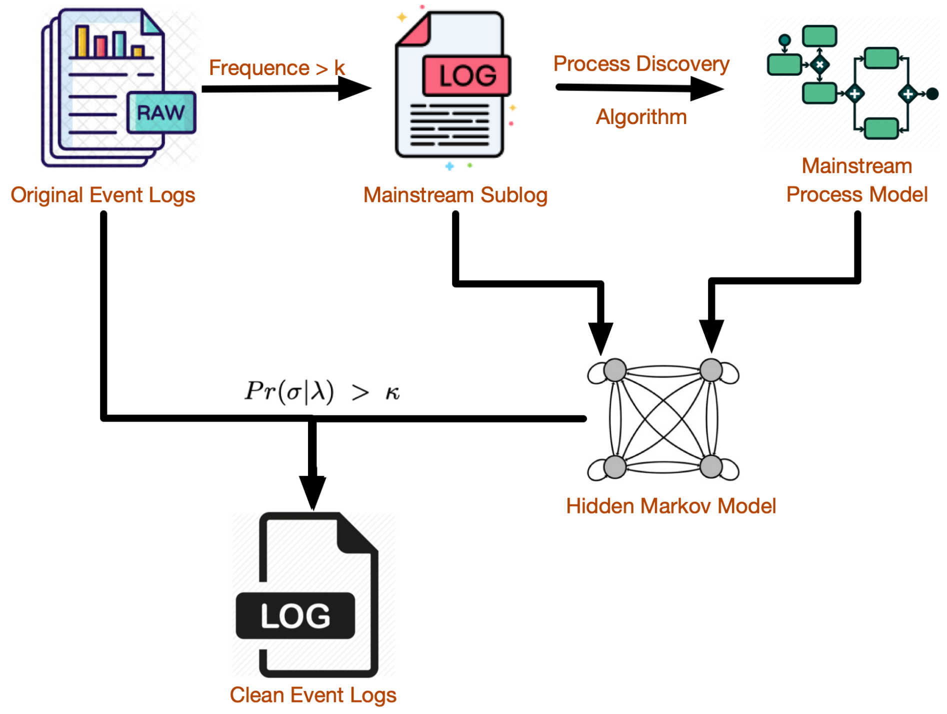


Figure 3.1: Workflow of using Hidden Markov Model to filtering out outliers from event logs

In [58], the authors propose a method for computing the occurrence likelihood of particular activities based on their preceding sequence of activities. In this method, an event with a likelihood lower than a pre-defined threshold is considered an outlier. Consequently, using trace-level filtering, the corresponding trace is opted out. The filtering method proposed in [30] yields an improvement upon the latter method by repairing traces that contain outlier event(s) instead of totally removing them. Outliers in this study are identified based on context frequency in traces. A drawback of this method is the addition of unreal activities to the event log through the repairing process.

All the algorithms mentioned above, based on direct event relationships, ignore the existence of parallel activities and long-term dependencies in the traces. In [58], the sequence length can be increased, but it increases the complexity of this method. Also, long-term relations cannot be identified in the proposed AFA algorithm. To address these issues and capture long term flow relationships between activities, [29] proposes a filtering technique based on sequential patterns and rules. e.g., whether a particular event is followed by another one somewhere in the trace or not. A limitation of this method is that it disregards the directly following relations and merely considers indirectly following relations. Therefore, this algorithm might fail to capture some outlier behaviors.

In this Chapter, we present an approach that uses a statistical model of sequential data, the hidden Markov model, to filter out outliers from event logs prior to applying any process discovery algorithms. Below we provide an outline of the approach, and an overview of the four steps required is shown in Figure. 3.1. First, we obtain the mainstream process model by applying an existing process discovery algorithm on process mainstream behaviors across the entire event log. For a given event log, the process mainstream behaviors imply either a set of frequently occurring event traces or a set of event traces that cover all of the frequently occurring activities. Next, we replay the original event log using the mainstream process model and obtain a mainstream sublog. The mainstream sublog shows us where the event traces fit within the mainstream process model. For the third step, we use the mainstream process model and the mainstream sublog to construct a hidden Markov model. Based on the hidden Markov model , we calculate the probability of the occurrence of each event trace, which is recorded in the original event log but not in the mainstream sublog, and compare it with a user-defined threshold. We define the event trace as an outlier if the probability of the occurrence of an event trace derived from the hidden Markov model is lower than the threshold. Finally, we remove the outliers from the original event logs to improve process discovery results.

The approach has been implemented on top of the *pm4py* Framework [7], and is evaluated by using the combination of two commonly used filtering approaches: the *Matrix Filter* approach [58] and the *Anomaly Free Automation* approach [20]. We measure the effectiveness of the proposed approach when using artificial and real-life event logs . The results of our experiments show that our approach adequately identifies and removes outliers, leading to an increase in the process discovery results. We measure these quality metrics in terms of fitness, precision, and F1-score. Additionally, we compare our process discovery results under different model complexity levels against the commonly used filtering approaches in terms of behavior recall, behavior precision, structural recall, and structural precision. The results of these experiments show that our approach outperforms two commonly used filtering approaches, namely the *Matrix Filter* approach and the *Anomaly Free Automation* approach.

This Chapter is organized as follows: we develop an approach to identify mainstream behaviors of the event log in Chapter 3.2. Chapter 3.3 presents the proposed approach to construct the hidden Markov model to filter out outliers from event logs. We evaluate the proposed outlier filtering technique in terms of four criteria mentioned above using artificial and real-life even logs and discuss our findings in Chapter 3.4.

3.2 Identify Mainstream Behaviors in Event Logs Based on Frequency-based Approaches

Now, we present a frequency-based approach to obtain the mainstream process model and the mainstream sublog for representing the primary process behaviors across the entire event log. Given an event log, the process mainstream behaviors imply either a set of event traces that occur frequently or a set of event traces that cover all of the frequently occurring activities. Our approach consists of three steps: (1) selecting a set of event traces based on frequency-based approaches; (2) applying an existing process mining algorithm on the set of event traces selected from (1) to obtain the mainstream process model; (3) obtaining the mainstream sublog by replaying the original event log on the mainstream process model.

To obtain the mainstream process model and the mainstream sublog that represents the mainstream process behaviors across a given event log, we convert the event log to the *grouped event log*. Before we give the *grouped event log* formal definition, we first introduce the *distinct activity sequence*.

Definition 8 (Distinct Activity Sequence)**.** *Given an event log* log *with an activity set Ω, the distinct activity sequence I is a finite sequence of activities $[t_1, \ldots, t_n]$ satisfying* $\exists \sigma \in \log, |\sigma| = |I| \wedge (\forall 0 < i < |\sigma|, act(\sigma_i) = I(i))$

For an illustrative purpose, we consider a simplified event log as shown in Table 3.1, the trace σ_1 is $[e_1, e_2, e_3, e_4]$ where $e_1 = (A, "08:15")$, $e_2 = (B, "10:24")$, $e_3 = (C, "10:25")$, $e_4 = (D, "13:19")$. The corresponding distinct activity sequence I_1 is $[A, B, C, D]$

Definition 9 (Grouped Event Log)**.** *Given an event log* log*, the grouped event log G corresponding to* log *is a set of distinct activity sequences $\{I_i\}$ satisfying $\exists I_1, I_2 \in G, (|I_1| = |I_2| \wedge \forall 0 < n < |I_1|, I_1(n) = I_2(n)) \rightarrow I_1 = I_2$. The superscript n of a distinct activity sequence indicates the number of corresponding event traces in the* log*. Labeling function $supp : I \mapsto \mathbb{N}$ is used to get the superscript of the distinct activity sequences.*

Consider the event log log in Table 3.1, for trace 1, 3, and 6. The corresponding distinct activity sequence for these traces is $[A, B, C, D]$. For trace 2 and trace 4, the corresponding distinct activity sequence is $[A, C, B, D]$. For the distinct activity sequence $[A, E, D]$, only one correspond event trace exists. Hence, the grouped event log G is $\{[A, B, C, D]^3, [A, C, B, D]^2, [A, E, D]\}$. The number of corresponding event traces has a

$supp([A, B, C, D]) = 3$. Note that, if the superscript n equals 1, we omit it for simplicity. Using this representation of an event log, we illustrate the steps to obtain the mainstream process model and the mainstream sublog for representing the process mainstream behaviors in the log.

Trace Identifier	**Activity**	**Timestamp**
Trace 1	A	08:15
Trace 2	A	08:24
Trace 3	A	09:30
Trace 1	B	10:24
Trace 3	B	10:24
Trace 2	C	10:26
Trace 1	C	10:25
Trace 4	A	11:45
Trace 2	B	11:46
Trace 2	D	12:23
Trace 5	A	13:14
Trace 4	C	13:17
Trace 1	D	13:19
Trace 6	A	13:39
Trace 3	C	14:09
Trace 6	B	14:19
Trace 3	D	14:29
Trace 4	B	14:43
Trace 5	E	15:22
Trace 6	C	15:29
Trace 5	D	15:45
Trace 4	D	16:10
Trace 6	D	16:43

Table 3.1: Example of event logs containing five activities and six event traces

In the first step, we use three different strategies to select the event traces to represent the following mainstream behaviors: 1.) frequent event traces, 2.) frequent activities, or 3.) a combination of both from the event log. For the frequent event traces, the event traces are selected from an event log based on the top xth frequency of the distinct activity sequences in the grouped event log, where x is user-defined. For instance, assuming that x equals two, the distinct activity sequences are the two most frequently occurring traces in

G: $[A, B, C, D], [A, C, B, D]$ and the corresponding set of event traces is $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_6\}$. For the frequent activities, we first calculate the frequency of each activity in the grouped event log. Then we obtain a set of frequent activities whose frequency is larger than a user-defined threshold. The event traces that are selected from the event log are based on the top frequencies of the distinct activity sequences that cover the frequent activities set. For instance, the frequencies of activities A, B, C, and D in the G are 3/11, 2/11, 2/11, 3/11, and 1/11 respectively. Assuming the threshold is 2, the set of frequent activities is $\{A, D\}$. The corresponding frequent distinct activity sequences, which cover the frequent activities set, are $[A, B, C, D]$ and the corresponding event traces in the log are $\{\sigma_1, \sigma_3, \sigma_6\}$. For the combination of frequent event traces and activities, we first pick up the event traces based on frequent activities. Then, from the remaining event traces, we select event traces based on the frequent event traces.

For the second step, we apply an existing process discovery algorithm to the event traces that are derived from the first step to obtain the mainstream process model. Finally, for event traces in the original event log, we identify whether an event trace is replayable defined below on the mainstream process model . We refer the set of replayable event traces as the mainstream sublog.

Definition 10 (Replayable)**.** *Given a workflow net $N = (P, T, F)$ and an event trace $\sigma = [e_1, \ldots, e_n]$, the σ is replayable on the N that satisfies $\forall j, 1 \le j \le n, (N, m^{start}) \xrightarrow{act(e_1)} \ldots \xrightarrow{act(e_j)} \ldots \xrightarrow{act(e_n)} (N, m^{end})$, where marking $m^{start} = (i, 1)$, and marking $m^{end} = (o, 1)$.*

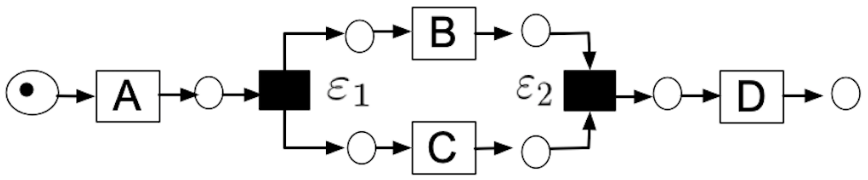


Figure 3.2: Mainstream process model corresponding to top frequency event traces

We use the following example to show how the mainstream sublog is obtained. Consider the event log log, and a mainstream process model $N = (P, T, F)$ shown in Figure 3.2 which is derived by applying an inductive miner algorithm [38] on the top two most frequent event traces. From the process model that is constructed by the inductive miner algorithm, there are activities (which are represented as rectangles) and invisible activities (which are represented as the black rectangles, such as ϵ_1 and ϵ_2) in Figure 3.2. The invisible activities and circles in the process model are only for routing purposes. These components are produced by process mining algorithms, but are not recorded in event logs.

For the event trace σ_1 shown in Table 3.1, the activity sequence is $[A, B, C, D]$. The initial marking of the mainstream process model $N = (P, T, F)$ is $\{(i, 1)\}$. According to formula 2.2, there is a firing sequence of transition $[A, B, C, D]$ in the process model, i.e., $(N, \{(i, 1)\}) \xrightarrow{A} (N, \{(P_1, 1)\}) \xrightarrow{\epsilon_1} (N, \{(P_2, 1), (P_3, 1)\}) \xrightarrow{B} (N, \{(P_3, 1), (P_4, 1)\}) \xrightarrow{C} (N, \{(P_4, 1), (P_5, 1)\}) \xrightarrow{\epsilon_2} (N, \{(P_6, 1)\}) \xrightarrow{D} (N, \{(o, 1)\})$. Based on definition 10, the event trace σ_1 is replayable on the N. Similarly, $\sigma_2, \sigma_3, \sigma_4$, and σ_6 are replayable on the mainstream process model. However, σ_5 is not replayable, since there is no firing sequence of transitions in the process model N. Hence, the mainstream sublog is $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_6\}$.

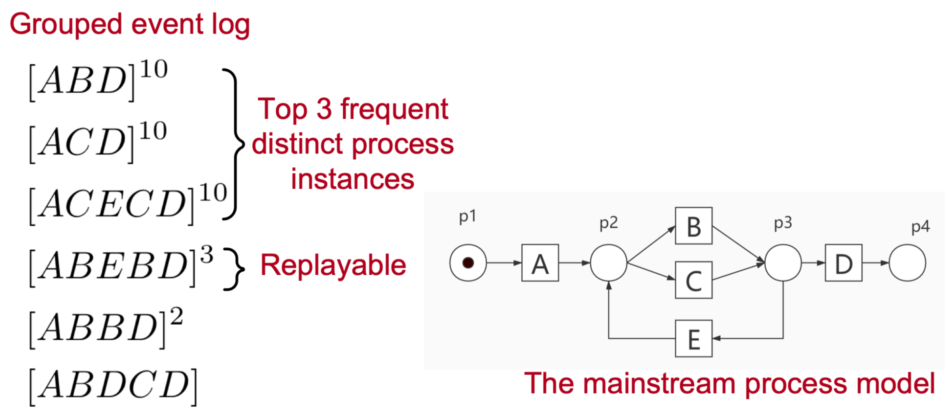


Figure 3.3: Mainstream process model corresponding the mainstream sublog

3.3 Construct Hidden Markov Models to Filter Out Outliers in Event Logs

In this Chapter, we first briefly introduce the hidden Markov Model definition. And then we present an approach to apply the hidden Markov model derived from the mainstream sublog and the mainstream process model to filter out outliers within the event logs. In particular, the first step of our approach is to construct a hidden Markov model based on the mainstream process model and the mainstream sublog. The second step is to apply the obtained hidden Markov model to identify the outliers and remove them from the event log.

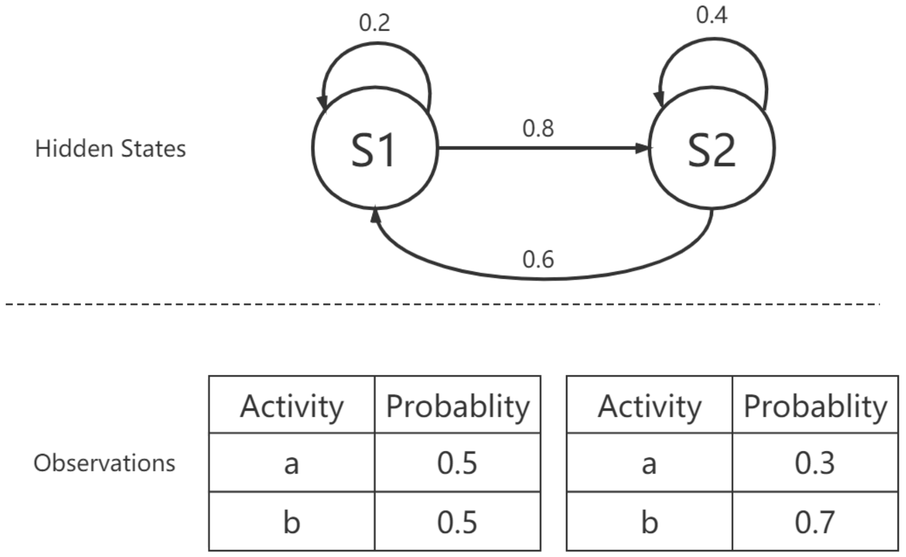
A hidden Markov Model is an extension of a discrete Markov process, which is a combination of two probability distribution processes wherein one of them is hidden. A hidden process can only be determined through another process that produces a sequence of observations [53]. Each observation depends on the states in a hidden process. The hidden Markov model can also be interpreted as Markov model wherein, its proper states are not directly observed [57]. The hidden Markov model is defined as follows.

Definition 11 (Hidden Markov Model [57])**.** *A Hidden Markov Model λ is a tuple* $(\mathbf{\Phi}, \mathbf{O}, \mathbf{A}, \mathbf{B}, \pi)$, *where:*

- $\mathbf{\Phi}$ *is a finite set of hidden states;*
- $\mathbf{O}$ *is a finite set of observations;*
- $\mathbf{A}$*:* $(\mathbf{\Phi} \times \mathbf{\Phi}) \rightarrow [0,1]$ *is a hidden state transition matrix, such that* $\forall_{s_1 \in \mathbf{\Phi}} \sum_{s_2 \in \mathbf{\Phi}} \mathbf{A}(s_1, s_2) = 1$*;*
- $\mathbf{B}$*:* $(\mathbf{\Phi} \times \mathbf{O}) \rightarrow [0,1]$ *are observation probabilities, such that* $\forall_{s \in \mathbf{\Phi}} \sum_{o \in \mathbf{O}} \mathbf{B}(s, o) = 1$*;*
- π*:* $\mathbf{\Phi} \rightarrow [0,1]$ *is the initial state distribution, such that* $\sum_{s \in \mathbf{\Phi}} \pi(s) = 1$.

From the above definition, we know that all of the observation elements could be produced in each of the hidden states. The probability of an observation $o \in \mathbf{O}$ in a hidden state $s \in \mathbf{\Phi}$ is denoted by the observation probability $\mathbf{B}(s, o)$. Note that, for a given event log log with an activity set Ω, we want to use a hidden Markov model to represent process mainstream behaviors observed in the event log log, and then link the observations that can be produced by the hidden Markov model to the activities in the event log by using the same identifier, i.e., $\mathbf{O} = \Omega$.

Given a hidden Markov model $\lambda = (\mathbf{\Phi}, \mathbf{O}, \mathbf{A}, \mathbf{B}, \pi)$ and an observation sequence $\Theta = o_1, o_2, \ldots, o_n$, the probability of the occurrence of Θ, i.e., $P(\Theta|\lambda)$, can be calculated by using the *Forward-Backward* [53, 12] algorithm. We apply this fundamental problem for Hidden Markov Models to identify whether an event trace is outlier in event logs.



Activity	Probablity
a	0.5
b	0.5

Activity	Probablity
a	0.3
b	0.7

Figure 3.4: Hidden Markov model example with $\mathbf{\Phi} = \{S_1, S_2\}, \mathbf{O} = \{A, B\}$, and $\pi = [1.0, 0.0]$

For an illustration purpose, we consider a simplified hidden Markov model $\lambda = (\mathbf{\Phi}, \mathbf{O}, \mathbf{A}, \mathbf{B}, \pi)$ shown in Figure 3.4. The depicted hidden Markov model has two hidden states $\mathbf{\Phi} = \{S_1, S_2\}$, two observations $\mathbf{O} = \{a, b\}$, and the initial state distribution is $\pi = [1.0, 0.0]$. The labeled arrows in this model correspond to the $\mathbf{A}$ matrix, i.e., $\mathbf{A} = \begin{bmatrix} 0.2 & 0.8 \\ 0.4 & 0.6 \end{bmatrix}$. In this example, the probability of a transition from hidden state S_1 to state S_2 is 0.8. The $\mathbf{B}$ matrix shown underneath the hidden state, i.e., $\mathbf{B} = \begin{bmatrix} 0.5 & 0.5 \\ 0.3 & 0.7 \end{bmatrix}$, gives the probability of producing each activity in that state. The probability of producing the activity a in hidden state S_1 is 0.5. Given an observations sequence $\Theta = aba$, the probability of occurrence of Θ corresponding to the λ is 0.1682.

While a hidden Markov model is an inherently stochastic model, a workflow net is an analytical representation and does not directly support a probabilistic description. Because of this, we need to infer the probabilistic parameters of a hidden Markov model from the mainstream sublog and the mainstream process model.

Given a grouped event log G that corresponds to the mainstream sublog and a mainstream process model $N = (P, T, F)$, we use following rules to construct the hidden Markov model $\lambda = (\mathbf{\Phi}, \mathbf{O}, \mathbf{A}, \mathbf{B}, \pi)$.

- Hidden states ($\mathbf{\Phi}$): each place in the mainstream process model is represented by exactly one state in the hidden Markov model λ, i.e. $|\mathbf{\Phi}| = |P|$;
- Observations ($\mathbf{O}$): the observations are the grouped event logs G that correspond to the mainstream sublog
- Hidden state transition matrix ($\mathbf{A}$): given two hidden state $s_1, s_2 \in \mathbf{\Phi}$, and the corresponding two places $p_1, p_2 \in P$, the element $\mathbf{A}(s_1, s_2)$ of $\mathbf{A}$ is

  $$\frac{\sum_{i\in[0,|G|]} sup(I_i)}{\sum_{i\in[0,|G|]} sup(I_i) \quad + \quad \sum_{j\in[0,|G|]} sup(I_j)}$$

  satisfying $\exists t_1 \in \bullet p_1, t_2 \in (p_1 \bullet \bigcap \bullet p_2) : [t_1, t_2] \in I_i$ and $\exists t_1 \in \bullet p_1, t_2 \in (p_1 \bullet \bigcap \bullet p_x) \wedge p_x \in (P - \{p_2\}) : [t_1, t_2] \in I_j$;

- Observation probabilities matrix ($\mathbf{B}$): given a hidden state $s \in \mathbf{\Phi}$, the corresponding place $p \in P$, and a transition $t \in T$, the element $\mathbf{B}(s,t)$ of $\mathbf{B}$ is assign as follows:

    1. When p is not in the output place, $\mathbf{B}(s,t) = \frac{\sum_{i \in [0,|G|]} sup(I_i)}{\sum_{j \in [0,|G|]} sup(I_j)}$ satisfying $\exists t_1 \in \bullet p : [t_1, t] \in I_i$ and $\exists t_1 \in \bullet p, t_2 \in p\bullet : [t_1, t_2] \in I_j$
    2. When p is the output place, the corresponding hidden state s is a final state, which does not produce any observable activity from the event log. Hence, the element $\mathbf{B}(s,t) = 0$. Instead, it is associated to a dummy end element ϵ, where the element $\mathbf{B}(s,\epsilon) = 1$.

- Initial state distribution (π): the probability of the hidden state corresponding to the input place in N equals one, and probabilities of all other hidden states equal zero.

In the following, we will use an example to illustrate the steps required to construct a hidden Markov model $\lambda = (\mathbf{\Phi}, \mathbf{O}, \mathbf{A}, \mathbf{B}, \pi)$ that is based on a mainstream sublog and a mainstream process model $N = (P, T, F)$.

Example 1. *Consider the new grouped event log G shown in Figure 3.3, we select the event traces corresponding to the top three frequency of the distinct activity sequences from the event log, i.e., $[ABD], [ACD]$, and $[ACECD]$. Then, we apply an existing process mining algorithm, for instance, the inductive miner algorithm developed by [38], on the selected event traces. The obtained mainstream process model is represented by a workflow net $N = (P, T, F)$ that is depicted in Figure 3.3. Based on the mainstream process model, the event traces corresponding to the distinct activity sequence $[ABEBD]$ are replayable. Hence, the grouped event log G that is corresponding to the mainstream sublog is $\{[ABD]^{10}, [ACD]^{10}, [ACECD]^{10}, [ABEBD]^{3}\}$.*

According to the rules to construct the hidden Markov model $\lambda = (\mathbf{\Phi}, \mathbf{O}, \mathbf{A}, \mathbf{B}, \pi)$, the hidden states $\mathbf{\Phi}$ contains s_1, s_2, s_3, s_4 that represent the places p_1, p_2, p_3, p_4 respectively. $\mathbf{O}$ is the grouped event log G that corresponds to the mainstream sublog, i.e., $\mathbf{O} = \{[ABD]^{10}, [ACD]^{10}$,

$[ACECD]^{10}, [ABEBD]^3\}$. *For initial state distribution* π*, the input place of* N *is* p_1*, which corresponds to the hidden state* s_1 *and* $\pi = [1, 0, 0, 0]$.

We take the hidden states s_3 *and* s_4 *as an example to illustrate the steps for element* $\mathbf{A}(s_3, s_4)$ *in a hidden state transition matrix. The places corresponding to the hidden states* s_3 *and* s_4 *are* p_3 *and* p_4*, respectively. As* $\bullet p_3 = \{B, C\}$ *and* $p_3 \bullet \bigcap \bullet p_4 = \{D\}$*, we sum up the superscripts of the distinct activity sequences containing the subsequent* $[B, D]$ *or* $[C, D]$*, such that the* $sup([ABD]) + sup([ACD]) + sup([ACECD]) + sup([ABEBD])$ *results in* 33 *as the numerator of* $\mathbf{A}(s_3, s_4)$*. Since* $p_3 \bullet \bigcap \bullet p_2 = \{E\}$*, we sum up the superscripts of the distinct activity sequences containing the subsequent* $[C, E]$ *or* $[D, E]$*, i.e., the* $sup([ACECD]) + sup([ABEBD])$ *results in* 13*. Hence the* $\mathbf{A}(s_3, s_4)$ *is* $\frac{33}{33+13} \approx 0.717$.

We take the hidden states s_2 *and* s_4 *as examples to illustrate different scenarios of constructing the observation probabilities matrix. The place corresponding to the hidden state* s_2 *is* p_2*. Consider the transition* C*, since* $\bullet p_2 = \{A\}$*, we sum up the superscripts of distinct activity sequences containing the subsequent* $[A, C]$*, i.e.,* $sup([ACD]) + sup([ACECD]) = 20$ *as the numerator for* $\mathbf{B}(s_2, C)$*. Since* $p_2 \bullet = \{B, C\}$*, we sum up the superscripts of distinct activity sequences containing the subsequent* $[A, B]$ *or* $[A, C]$*, i.e.,* $sup([ACD]) + sup([ABD]) + sup([ABEBD]) + sup([ACECD])$*, and the following result is* 33*. Hence the* $\mathbf{B}(s_2, C)$ *is* $\frac{20}{33} \approx 0.606$*. For the hidden state* s_4*, the place* p_4 *corresponding to* s_4 *is output place in the mainstream process model. According to the rule, the element* $\mathbf{B}(s_4, A) = \mathbf{B}(s_4, B) = \mathbf{B}(s_4, C) = \mathbf{B}(s_4, D) = \mathbf{B}(s_4, E) = 0$*, and* $\mathbf{B}(s_4, \epsilon) = 1$*. Similarly, based on the rules, we construct other elements of* $\mathbf{A}$ *and* $\mathbf{B}$*, and get the following results:*

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0.28 & 0 & 0.72 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.39 & 0.61 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.28 & 0.72 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In the second step, we take the event log log and use the hidden Markov model λ produced by the first step as input. Given an event trace $\sigma = [e_1, \ldots, e_n]$, we compute the probability of σ, given the model λ, i.e., $Pr(\sigma|\lambda)$, using the *Forward-Backward Procedure* [12, 11]. Then, we compare the resulted probability with the user-defined threshold κ. We identify each event trace in the log with the probability conditioned on λ being higher than the threshold, i.e., $Pr(\sigma|\lambda) > \kappa$ as an outlier. Finally, we remove the outliers from the event log in order to improve process discovery results. For example, consider the event trace corresponding to the distinct activity sequence $[ABDCD]$ in Example 1. We apply the *Forward-Backward Procedure* to the event trace, and the probability is 0.038. Assuming the threshold of κ is 0.01, the event trace corresponding to $[ABDCD]$ is not an outlier. Similarly, we calculate the probability of the event trace corresponding to $[ABBD]$ and find that it is under the λ, so we treat it as an outlier. Finally, the resulted event log contains event traces corresponding to distinct activity sequence $[ABD], [ACD], [ACECD], [ABEBD], [ABDCD]$.

3.4 Experimental Evaluation

In this Chapter, we evaluate our proposed approach using both artificial event logs and real-life event logs from [1, 2, 3, 5, 4]. The artificial logs are generated from the Processes and Logs Generator (Plg) [18], which is an open-source tool that is used to generate artificial event logs based on the user-defined process model. The characteristics of the real-life event logs we selected are summarized in Table 3.2. The objectives of our evaluations consist of two components. First, we evaluate if filtering out outliers by using the hidden Markov model can improve the quality of the process models that are discovered from event logs. To do so, we choose an existing Inductive mining algorithm [38], and apply it on the same real-life event logs with and without the hidden Markov model approach. Second, we compare the performance of the proposed approach with two commonly used filtering approaches, i.e., the *Matrix Filter* approach [58] and the *Anomaly Free Automation* approach [20], using the artificial and real-life event logs. Before the evaluation, we first define our evaluation criteria.

DataSet	Number of Traces	Number of Event Entry	Average number of Event Entry
BPIC2013 [1]	819	2351	2.871
BPIC2020 [2]	10500	56,437	5.374
Hospital Billing [3]	100000	451359	4.514
Road Traffic [4]	150370	561470	3.734

Table 3.2: Characteristics of event logs used for evaluations in the cleaning phase

3.4.1 Performance Metrics

The quality of a process model is evaluated by two criteria: (1) fitness f, and (2) precision p. The *fitness* measures how well a model can reproduce the process behavior that is contained within a log, and the *precision* measures the degree to which the behavior that is made possible by a model is found within a log [20]. A model with good Fitness allows for behavior seen in the log. This means that if a model has perfect fitness, then all of the behavior seen in the log is shown in the provided process model. Similar, the higher precision values are, the better quality of the process model. We use the pm4py [8] library to calculate these performance metrics value.

Definition 12. *(Fitness [56]) Let k be the number of different traces from the aggregated log. For each log trace i ($1 \leq i \leq k$), n_i is the number of process instances combined into the* *current trace, m_i is the number of missing tokens, r_i is the number of remaining tokens, c_i is the number of consumed tokens, and p_i is the number of produced tokens during log replay of the current trace. The fitness metric f is defined as follows:*

$$f = \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} n_i m_i}{\sum_{i=1}^{k} n_i c_i}\right) + \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} n_i r_i}{\sum_{i=1}^{k} n_i p_i}\right)$$

Definition 13. *(Precision [56]) Let k be the number of different traces from the aggregated log. For each log trace i ($1 \leq i \leq k$), n_i is the number of process instances combined into the current trace, and x_i is the mean number of enabled transitions during log replay of the current trace (note that invisible tasks may enable succeeding labeled tasks but they are not counted themselves). Furthermore, T_V is the set of visible tasks in the Petri net model. The precision metric p is defined as follows:*

$$p = \frac{\sum_{i=1}^{k} n_i(|T_V| - x_i)}{(|T_V| - 1) \cdot \sum_{i=1}^{k} n_i}$$

However, the fitness and precision are two aspects of a process model which may not always be consistent. The $F1$ score [69] is defined as the harmonic mean of the weighted average fitness f^W and precision p^W, i.e. $F1 = \frac{2 \times f^W \times p^W}{f^W + p^W}$. We will also use the $F1$ score as an evaluation criteria.

For any given artificial event log, we will have two process models that represent the reference model and the mined model. As such, we need to determine the behavioral and structural similarity between these two models [24]. The behavioral similarity metrics analyze the event log to quantify how similar the behavior of the mined model is to that of its reference model in terms of precision and recall [19]. The higher that the behavioral precision and recall is, the greater the similarity is between the referenced model and the mined model. The structural similarity metrics reflects the degree of correct causality relations that exist in the referenced model or the mined model. This is typically measured in terms of precision and recall. A value of both structural precision and recall close to 1 indicates two process models are structurally, very similar.

Definition 14. *(Behavioral precision and recall [19]) Let σ be a trace in an event log. $L(\sigma)$ be the number of occurrences of σ in an event log. N_r and N_m be the respective Petri net for the reference and the mined models. C_r and C_m be the respective causality relations for N_r and N_m. The behavioral precision $B_p$and recall B_r is defined as:*

$$B_p = \frac{\sum_{\sigma \in L}\left(\frac{L(\sigma)}{|\sigma|} \times \sum_{i=0}^{|\sigma|-1} \frac{Enabled(C_r,\sigma,i) \cap Enabled(C_m,\sigma,i))}{Enabled(C_m,\sigma,i)}\right)}{\sum_{\sigma \in L} L(\sigma)}$$
$$B_r = \frac{\sum_{\sigma \in L}\left(\frac{L(\sigma)}{|\sigma|} \times \sum_{i=0}^{|\sigma|-1} \frac{Enabled(C_r,\sigma,i) \cap Enabled(C_m,\sigma,i))}{Enabled(C_r,\sigma,i)}\right)}{\sum_{\sigma \in L} L(\sigma)}$$

Definition 15. *(Structural precision and recall [19]) Let C_r and C_m be the respective causality relations for N_r and N_m. The structural precision S_p and structural recall S_r are defined as:*

$$S_p = \frac{|C_r \cap C_m|}{|C_m|} \; S_r = \frac{|C_r \cap C_m|}{|C_r|}$$

3.4.2 Process Model Discovery Performance Improvement Using the Hidden Markov Model

This set of experiments is to evaluate whether filtering out outliers by using the hidden Markov model can improve the quality of process models discovered from event logs. More specifically, given an event log, we apply the three different approaches presented in Chapter 3.2 to obtain the mainstream sublog and mainstream process model. For each mainstream sublog and mainstream process model , we apply the proposed approach in Chapter 3.3 to construct the hidden Markov model and filter out outliers from the original event log. Then, we discover a process model from event logs using the inductive mining algorithm presented in [38]. The *fitness*, *precision*, and *F1-score* of each event log and their respective process models are depicted in Table 3.3.

	BPIC 2013			BPIC 2020			Road Traffic			Hospital Billing		
	Fitness	Precision	F1	Fitness	Precision	F1	Fitness	Precision	F1	Fitness	Precision	F1
No Filtering Approach	0.922	0.478	0.629	0.913	0.679	0.778	0.768	0.561	0.648	0.785	0.539	0.639
Activity Approach and HMMs	0.935	0.595	0.798	0.945	0.785	0.858	0.791	0.604	0.684	0.858	0.543	0.665
Trace Approach and HMMs	0.937	0.577	0.786	0.947	0.787	0.860	0.812	0.667	0.732	0.826	0.597	0.693
Combination and HMMs	0.947	0.641	0.832	0.958	0.842	0.896	0.829	0.709	0.764	0.889	0.61	0.723

Table 3.3: *Fitness*, *precision*, and *F1-score* of each process models from different event logs

From the experiment results, it is clear that when we use our proposed approach to filter out the outliers from an event log, prior to applying process discovery algorithms, we are able to achieve higher fitness, precision, and F1 score with different real-life event logs. For the real-life event logs, the average improvement percentage of fitness, precision, and F1 score are 7.23%, 24.44%, and 19.57%, respectively.

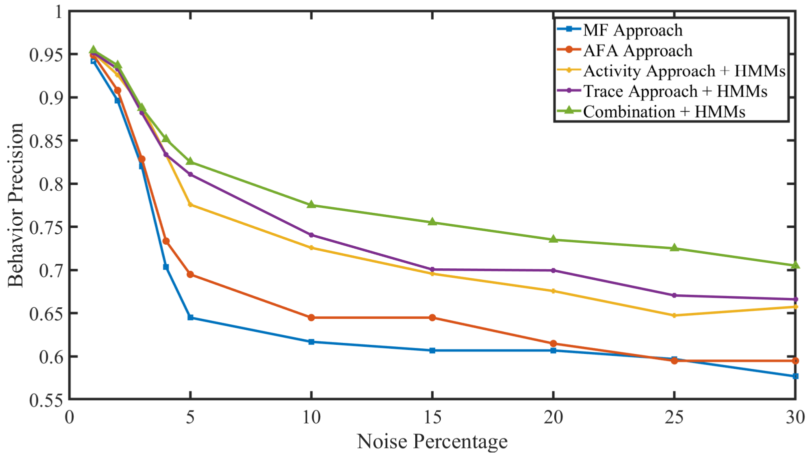


Figure 3.5: Behavior precision generated from event logs with different noise levels

3.4.3 Hidden Markov Model Approach Vs Two Commonly Used Filtering Approaches

This set of experiments is to compare our proposed approach with the two commonly used filtering approaches such as the *Matrix Filter* approach (**MF**) [58] and the *Anomaly Free Automation* approach (**AFA**) [20] using the aforementioned artificial and real-life event logs.

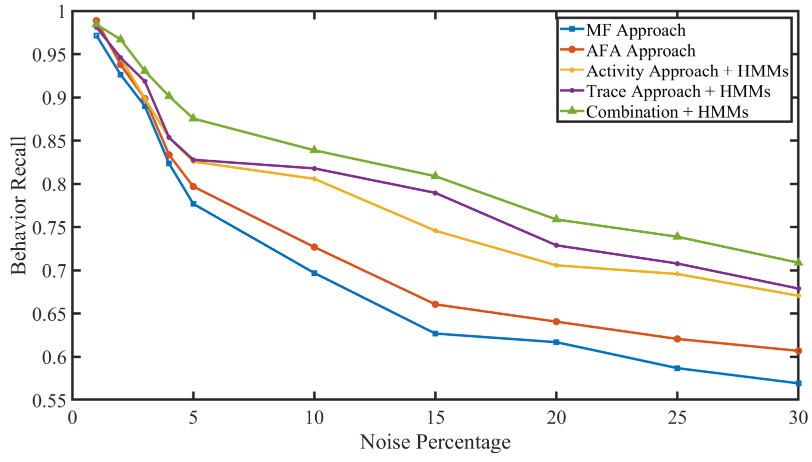


Figure 3.6: Behavior recall generated from event logs with different noise levels

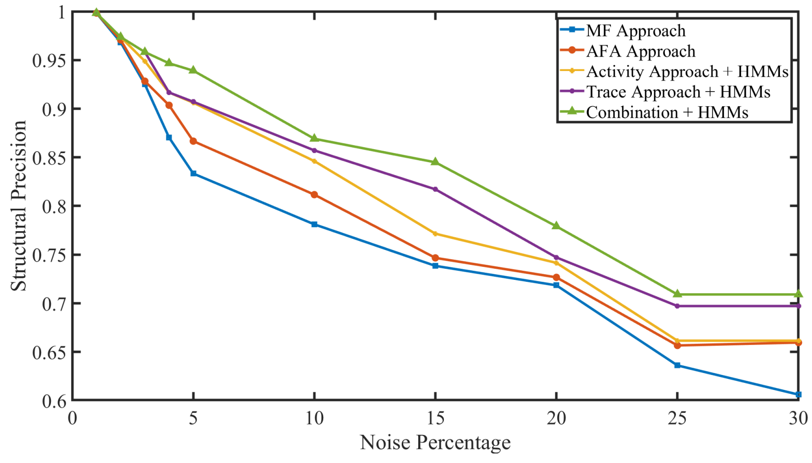


Figure 3.7: Structural precision generated from event logs with different noise levels

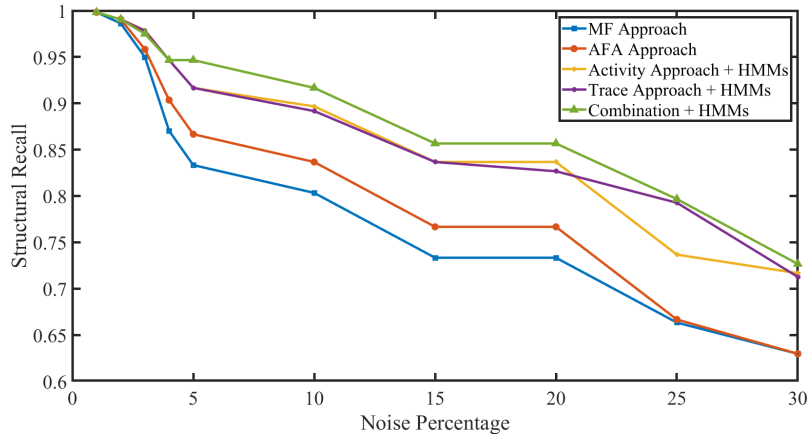


Figure 3.8: Structural recall generated from event logs with different noise levels

Artificial event logs: For benchmarking the different filtering approaches, we consider two different complexity levels for the reference models by using four basic structures found within a process. The first level of the reference model contains a parallel structure, an exclusive-choice structure, a loop structure, and a sequence structure. The second level of the reference model contains the interactions between these basic structures by combining them together. For each level of the reference model, we create ten different reference models that contain an increasing activity number. Then, for each referenced model, we use the *Plg* tool to generate ten event logs by injecting an incremental amount of noise ranging from 1% to 30% to produce an event log that contains one thousand event traces.

For each artificial event log, we first use the three different approaches to obtain the mainstream sublog and mainstream process model , and construct the hidden Markov models. Then, we filter out the outliers from the given event log based on the hidden Markov model. We also apply the two commonly used approaches, i.e., **MF** and **AFA** on the given artificial event log to filter out the outliers. For each clean event log derived from five different filtering approaches , we apply the inductive mining algorithm to discover a process model, and calculate the behavioral recall, behavioral precision, structural recall, and structural precision. The results are depicted in Figure 3.5, Figure 3.6, Figure 3.7, , Figure 3.8.

As shown in results, the proposed approach outperforms the two commonly used approaches on average cases. More specifically, the performance of our proposed approach (Combination + HMMs) is 10.43%, 11.92%, 9.61%, 9.20% higher than the performance of the **MF** approach with respect to behavioral recall, behavioral precision, structural recall ,and structural precision, respectively. The performance of our proposed approach (Combination + HMMs) is 8.89%, 9.57%, 7.47%, 8.85% higher than the performance of **ANA** approach with respect to behavior recall, behavior precision, structural recall ,and structural precision, respectively. We also observe that the approach using the combination strategy to construct the hidden Markov models outperforms the approaches using frequent activity or event traces to construct a hidden Markov models with respect to the four criteria.

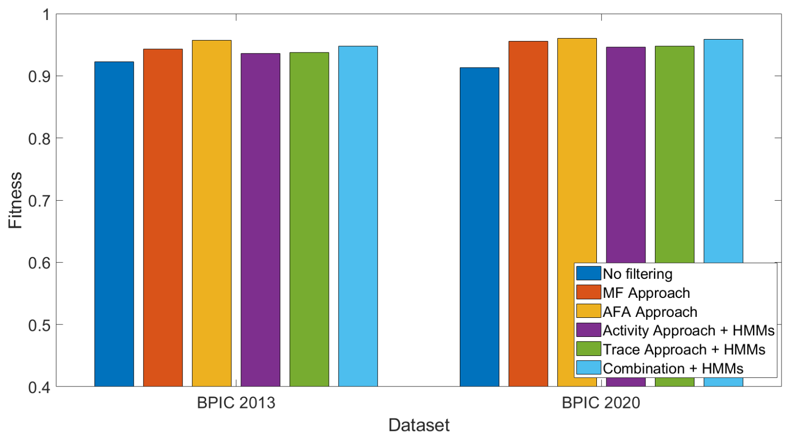


Figure 3.9: Fitness values of each process models from real life event logs

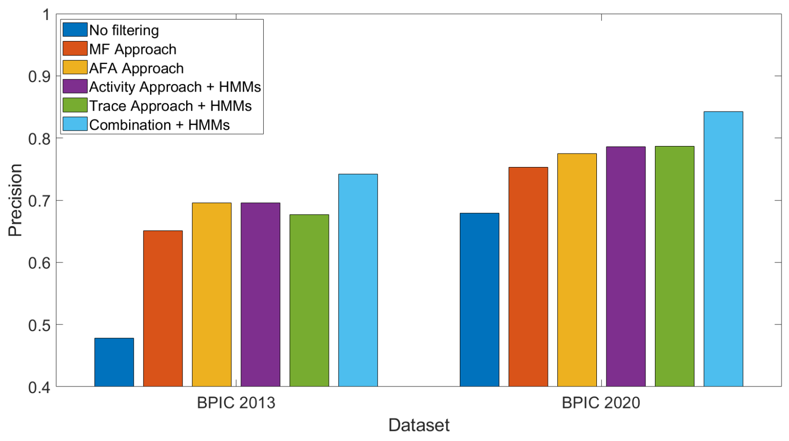


Figure 3.10: Precision values of each process models from real life event logs

Real-life event logs: We use five different approaches, i.e., **MF** approach, **AFA** approach, HMMs derived from frequent activity approach, HMMs derived from frequent event trace approach, and HMMs derived from combination approach on the real-life event logs to filter out outliers. Then, we apply the inductive mining algorithm on clean event logs to discover a process model and calculate fitness, precision, and F1 score. The results are depicted in Figure 3.9, Figure 3.10, and Figure 3.11.

From the experiment results, the performance of the proposed approach (Combination + HMMs) is 0.4%, 11.3%, and 6.7% higher than the **MF** approach with respectively for *fitness*,

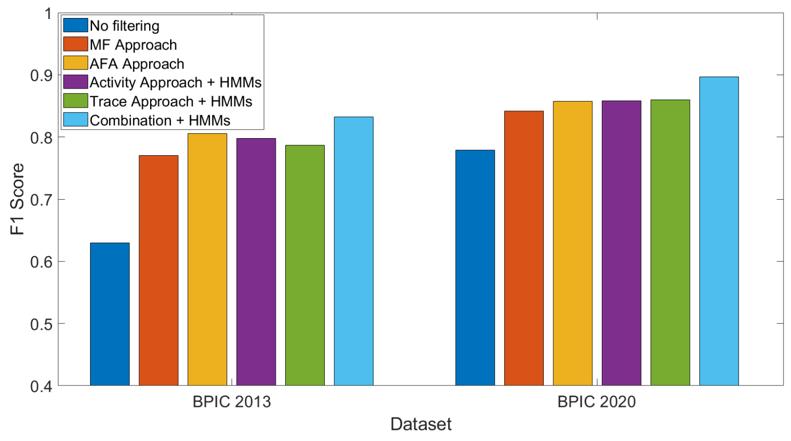


Figure 3.11: F1 Scores of each process models from real life event logs

precision, and *F1 score*. The fitness derived from the proposed approach (Combination + HMMs) is 0.54% lower than the **ANA** approach, but the precision is higher. Hence, we are able to achieve 3.9% higher F1 score with different real-life event logs.

3.5 Summary

In this Chapter, we present an approach that uses hidden Markov models to filter out outliers from event logs prior to applying process discovery algorithms to improve process discovery results. Our experiments on artificial event logs and real-life event logs show that: (1) process models obtained by filtering out outliers with hidden Markov models have higher fitness, precision, and F1 score than process models obtained by directly applying discovery algorithms on the event logs; (2) the proposed filtering method outperforms two commonly used filtering approaches, namely the *Matrix Filter* approach and the *Anomaly Free Automation* approach for both artificial event logs and real-life event logs.

Chapter 4

Implicit Knowledge Discovery - Timing Constraints

4.1 Background and Related Work

Once the raw event logs are cleaned by the approach presented in Chapter 3, the next step is to discover implicit knowledge from a clean event log. As we mentioned above, most process mining algorithms focus only on the functional aspects of a process, the timing information as to *when* an action must take place is neglected. However, the timing information can be critical in many domains, such as patient care. For instance, an ionized calcium test must be tested within 10 minutes of its collection, and blood specimens received in a lab after 10 minutes need to be re-collected for accurate results. These timing constraints need to be enforced in order to provide safe and effective patient care. However, process models uncovered by most existing process mining techniques only reveal the underline structures of an actual process, the timing constraints, such as the 10-minute limit in the ionized calcium test example, are not reflected in the process models

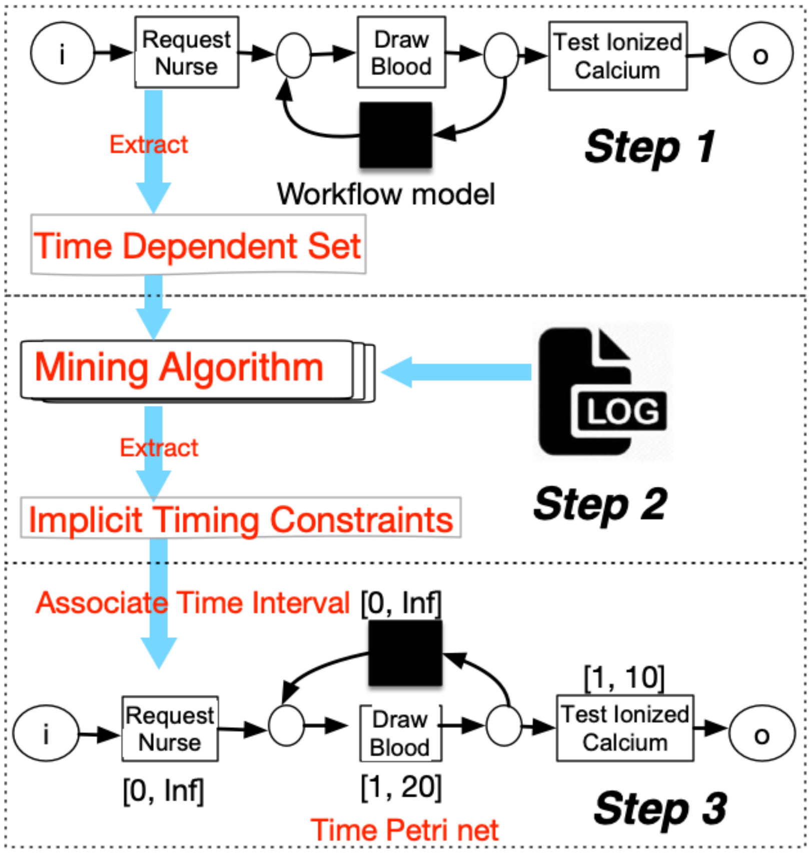


Figure 4.1: Mining and integrating timing constraints in workflow models

This chapter presents an approach that extends existing process mining techniques to discover and integrates timing constraints with a process model constructed by any existing process mining algorithm. The approach contains three major steps, i.e., first for a given workflow model constructed by an existing process mining algorithm and represented by a workflow net (which belongs to a subset of Petri net) and, extract a time dependent set associated with each transition in the workflow model. Second, based on the time dependent sets, develop an algorithm to extract implicit timing constraints from event logs for each transition in the model. Third, extend the original workflow model into a time Petri net where the discovered timing constraints are associated with their corresponding transitions. Figure 4.1 depicts the architecture of our approach. A real-life road traffic fine management process [49] is used as a case study to investigate the effectiveness and validity of the approach. The case study provides evidence that the approach is able to discover timing constraints of an actual workflow process from event logs.

This Chapter is organized as follows: we present an approach to extract time dependent set in Chapter 4.2. Chapter 4.3 develops the algorithm to mine the timing constraints from event logs based on time dependent sets and integrate the timing constraints into the workflow net in the form of the time Petri net. A real-life road traffic fine management process scenario is performed in Chapter 4.4 to investigate the effectiveness and validity of the approach.

4.2 Extract Time Dependent Set from a Workflow Net

Before we show how to extract a time dependent set of a transition, we first give its formal definition.

Definition 16 (Time Dependent Set). *Given a workflow net $N = (P, T, F)$, where $T = A \cup \Gamma$, $A \cap \Gamma = \emptyset$. A is a set of non-invisible transitions, and Γ is a set of invisible transitions. The time dependent relation set $\Theta(t)$ of a transition $t \in A$ is defined as follows:*

$$\Theta(t) = \{a \in A | (a\bullet \cap \bullet t \neq \emptyset) \vee (\exists \varepsilon_1, ..., \varepsilon_i \in \Gamma : a\bullet \cap \bullet\varepsilon_1 \neq \emptyset \wedge \varepsilon_{i-1}\bullet \cap \bullet\varepsilon_i \neq \emptyset \wedge \varepsilon_i\bullet \cap \bullet t \neq \emptyset)\}$$

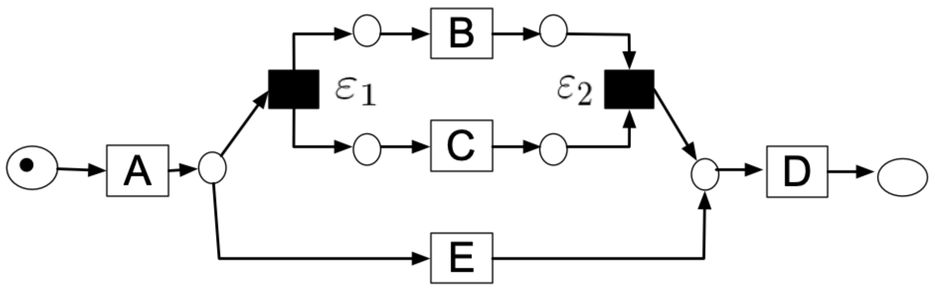


Figure 4.2: Process model corresponding to the event log shown in Table 3.1

Example 2. *Consider the event log given in Table 3.1 and a corresponding process model shown in Figure 4.2 which is derived by applying an inductive miner algorithm [14] on the given log. In the process model constructed by the inductive miner algorithm, there are* *activities represented in rectangles and invisible activities represented in black rectangles, such as ε_1 and ε_2, in Figure 4.2. Invisible activities and circles in the process model are only for routing purposes. They are produced by process mining algorithms, but not recorded in event logs.*

For activity D's time dependent set, as the invisible activity ε_2 is connected to the target activity D, we need to trace back until we find a visible activity that connects to the invisible activity ε_2, which are activity B and C. Hence, activity D's time dependent set $\Theta(D) = \{E, B, C\}$. Similarly, the time dependent sets of the activity A, B, C, and D are $\Theta(A) = \emptyset, \Theta(B) = \{A\}, \Theta(C) = \{A\}$, and $\Theta(E) = \{A\}$, respectively.

Algorithm 1 gives the procedure of extracting time dependent set of the transition x in the workflow net. The time complexity of algorithm 1 is $O(N^3)$, where N is the number of transitions in the workflow net.

4.3 Mine Timing Constraints from Event logs

We design an algorithm to automatically mining timing constraints for every transition based on the time dependent set and represent the workflow model associated with timing constraints by time Petri nets.

Given a workflow $N = (P, T, F)$ mined from an event log log, we first construct a time Petri net $N' = (P, T, F, I)$ where $\forall t \in T : I(t) = [0, +\infty]$. The constructed time Petri net N' has the same execution semantics as N as explained in Chapter 2.2. The next step is to mine the firing time interval for every transition t from event logs and update the corresponding $I(t)$.

As presented in Chapter 4.2, the time dependent set $\Theta(t)$ of a transition t contains all

Algorithm 1 Extracting Time Dependent Set of a Transition

```
Require: a workflow net $N = (P, T, F)$, a transition $x$
Ensure: The time dependent set $\Theta$ of transition $x$
 1: Define three sets $\Theta(x) = \{\}$ ; $A = \{\}$ ; $\Gamma = \{\}$
 2: for each transitions $t \in T$ do
 3:    if  $t$ is non-invisible transition then
 4:       $A = A \cup \{t\}$
 5:    else
 6:       $\Gamma = \Gamma \cup \{t\}$
 7:    end if
 8: end for
 9: for each transition $t_0 \in T$ do
10:    if  $\bullet x \cap t_0 \bullet \neq \emptyset$ then
11:       $\Theta(x) = \Theta(x) \cup \{t_0\}$
12:    end if
13: end for
14: repeat
15:    Define break condition: $condition = False$
16:    for each transition $t_1 \in \Theta(x)$ do
17:       if $t_1 \in \Gamma$ then
18:          $condition = True$
19:          $\Theta(x) = \Theta(x) - \{t_1\}$
20:          for each transition $t_2 \in T$ do
21:             if  $\bullet t_1 \cap t_2 \bullet \neq \emptyset$ then
22:                $\Theta(x) = \Theta(x) \cup \{t_2\}$
23:             end if
24:          end for
25:       end if
26:    end for
27: until $condition$
28: return  $\Theta_x$
```

transitions that have immediate timing impact on the transition t. We use the following rules to determine the timing constraints for each transition in a workflow net from the workflow net's corresponding event logs: if a transition directly follows the workflow net's input place, i.e., $\Theta(t) = \emptyset$, we set its timing constraint as $I(t) = [0, +\infty]$; otherwise, $I(t) = [a, b]$ where a and b are determined as following:

- **Earliest Firing Time:** the transition t's earliest firing time (a) is the minimum value of time between an occurrence of t and its most recent occurrence of any transition

$t' \in \Theta(t)$ that occurs before t in the same event trace, i.e., $a = \min\{(\tau_c - \tau_r) | e_c = (t, \tau_c) \wedge e_r = (t_r, \tau_r) \wedge \Phi(e_c, e_r) = \texttt{True}\}$, where

$$\begin{aligned} \Phi(e_c, e_r) = & \forall \sigma \in \log : \forall e_i(t_i, \tau_i) \in \sigma : t_i = t \wedge \\ & \tau_c = \tau_i \wedge (\exists e_j(t_j, \tau_j) \in \sigma : 0 < j < i \wedge \\ & t_j \in \Theta(t) \wedge \tau_j = \tau_r \wedge (\forall e_k(t_k, \tau_k) \in \sigma : \\ & j < k < i \wedge t_k \notin \Theta(t))) \end{aligned} \tag{4.1}$$

- **Latest Firing Time:** the transition t's latest firing time (b) is the maximum value of time between an occurrence of t and its most recent occurrence of any transition $t' \in \Theta(t)$ that occurs before t in the same event trace, i.e., $b = \max\{(\tau_c - \tau_r) | e_c = (t, \tau_c) \wedge e_r = (t_r, \tau_r) \wedge \Phi(e_c, e_r) = \texttt{True}\}$, where $\Phi(e_c, e_r)$ is given in formula (4.1).

Theorem 1. *Given an event log* $\log = \{\sigma | \sigma = e_1(t_1, \tau_1), \ldots, e_i(t_i, \tau_i), \ldots, e_n(t_n, \tau_n)\}$ *and corresponding time Petri net* $N' = (P, T, F, I)$ *with* $I(t) = [a, b]$ *derived by the mining rules for each* $t \in T$*, all timestamps of corresponding activity* t *in log* log *satisfy the timing constraint* $I(t) = [a, b]$.

Proof. We use contradiction to prove the theorem. For transition t, suppose there exist event entries $e = (t, \tau)$ and $e' = (t', \tau')$ in the same event trace σ, and the transition t's timing constraint $\mathcal{C}(t)$ determined by event entries e and e' is $\tau - \tau' < a$. Because the event entries e and e' determine the timing constraint $\mathcal{C}(t)$ for t, $\Phi(e, e') = \texttt{True}$ holds, where $\Phi(e, e')$ is defined in formula (4.1). The timing constraint $\mathcal{C}(t) < a$ contradicts the **Earliest Firing Time** rule, i.e., $a = \min\{(\tau_c - \tau_r) | e_c = (t, \tau_c) \wedge e_r = (t_r, \tau_r) \wedge \Phi(e_c, e_r) = \texttt{True}\}$. Hence, transition $t's$ timing constraint must be larger than or equal to a, i.e., $\mathcal{C}(t) \geq a$.

Similarly, suppose there exist event entries $e = (t, \tau)$ and $e' = (t', \tau')$ which determines the transition t's timing constraint $\mathcal{C}(t)$ as $\tau - \tau' > b$. Since the event entries e and e'

determine the timing constraint $\mathcal{C}(t)$, $\Phi(e, e') = \texttt{True}$ holds, where $\Phi(e, e')$ is defined in formula (4.1). The timing constraint $\mathcal{C}(t) > b$ contradicts the **Latest Firing Time** rule, i.e., $b = \max\{(\tau_c - \tau_r) | e_c = (t, \tau_c) \wedge e_r = (t_r, \tau_r) \wedge \Phi(e_c, e_r) = \texttt{True}\}$. Hence, transition $t's$ timing constraint must be smaller than or equal to b, i.e., $\mathcal{C}(t) \leq b$.

Therefore, $a \leq \mathcal{C}(t) \leq b$ holds, i.e., all timestamps of activity t in event log log satisfy the timing constraint interval $I(t) = [a, b]$. □

It is worth pointing out that the timing constraints defined by these rules may not be tight. We will give an example to explain a possible reason for its looseness in Chapter 4.4.

Algorithm 2 gives the detailed steps of applying the rules to automatically mine and update the firing timing constraints for every transition in a workflow net model. The time complexity of Algorithm 2 is $O(K \times M \times N^2)$, where K is the number of transitions in the process model, M is the number of traces in the event log, and N is the number of events in a trace. We illustrate Algorithm 2 in Example 3 with the event log given in Table 3.1.

Example 3. *The Petri net N mined from the the event log* log *in Table 3.1 is depicted in Figure 4.2. We first construct a time Petri net $N' = (P, T, F, I)$ where $\forall t \in T : I(t) = [0, +\infty]$. According to Algorithm 1, the time dependent set of N' is $\Theta = \{\Theta(A) = \emptyset, \Theta(B) = \{A\}, \Theta(C) = \{A\}, \Theta(D) = \{B, C, E\}, \Theta(E) = \{A\}\}$.*

We take transitions A and D as examples to illustrate different scenarios of applying Algorithm 2. As $\Theta(A) = \emptyset$, based on Lines 2-4, Algorithm 2 does not update the firing time interval of transition A, i.e., $I(A) = [0, +\infty]$.

The time dependent set of transition D is $\Theta(D) = \{B, C, E\}$. We apply Lines 7-16 of Algorithm 2 for every trace in the event log log*. For instance, the trace $\sigma_1 = \{A, B, C, D\}$ only contains one occurrence of event D at time "08/05/2019 : 13:19". Based on Lines 9-15, we locate the event C occurring at "08/05/2019 : 10:25" and calculate the following results*

Algorithm 2 Mining Timing Constraints

```
Require: The constructed time Petri net $N' = (P, T, F, I)$, the corresponding event log
    $log = \{\sigma\}$, and the corresponding time dependent set $\Theta = \{\Theta(t) | t \in T\}$.
Ensure: The time Petri net $N' = (P, T, F, I)$ with $I$ updated.
 1: for each $t \in T$ do
 2:   if $\Theta(t) = \emptyset$ then
 3:     CONTINUE
 4:   end if
 5:   Define temporary variables $a = +\infty$ and $b = -\infty$
 6:   for each $\sigma = e_1(t_1, \tau_1), \ldots, e_i(t_i, \tau_i), \ldots, e_n(t_n, \tau_n) \in log$ do
 7:     for each $1 \leq i \leq n$ do
 8:       if $t_i = t$ then
 9:         $j = i - 1$
10:         while $j \geq 1$ do
11:           if $t_j \in \Theta(t)$ then
12:             $\Delta = \tau_i - \tau_j$
13:             $a = \min(a, \Delta)$, $b = \max(b, \Delta)$
14:             BREAK
15:           end if
16:           $j--$
17:         end while
18:       end if
19:     end for
20:   end for
21:   $I(t) = [a, b]$
22: end for
```

$\Delta = 174$*,* $a = 174$*, and* $b = 174$ *with minutes as the time units. After applying the procedure to other four traces, the final results are* $a = 20$ *and* $b = 174$*, i.e.,* $I(D) = [20, 174]$.

Similarly, we apply Algorithm 2 to other three transitions and get the following results $I(B) = [54, 202]$*,* $I(C) = [92, 339]$*, and* $I(E) = [128, 128]$*. The updated time Petri net is shown in Figure 4.3.*

The proposed techniques of mining timing constraints from event logs for a given workflow net model is implemented on an existing open-source process mining framework *pm4py* [8]. The implementation of the technique is available at `https://github.com/stevenzzy9/Wastewater-Treatment-Process-Discovery`.

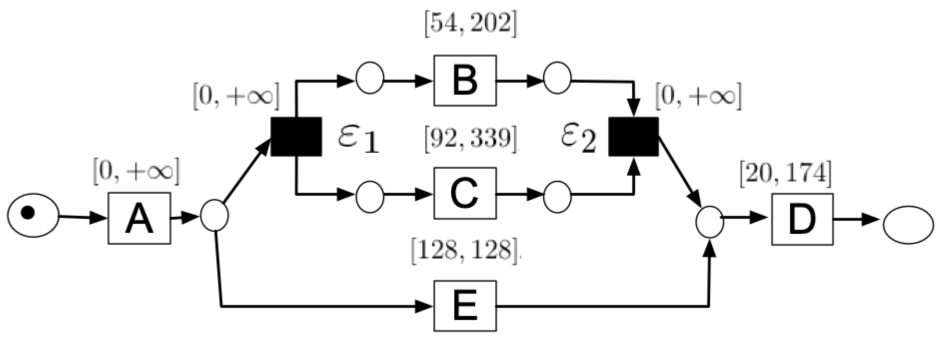


Figure 4.3: Process model integrated with timing constraints

4.4 Road Traffic Fine Management Process Case Study

In this Chapter, we apply the proposed approach to a road traffic fine management process used in Italy. By the Italian law, a traffic fine management process starts with the *Create Fine* activity. A fine notification must be sent within 90 days since its creation. After being notified, the offender can either pay the fine or appeal the fine within 60 days of notification. If the fine is paid, the corresponding case is closed. The system maintains a case for up to 5 years from the corresponding offense is committed. The snapshot of the event log was taken in June 2013 [6]. After filtering out all open (unclosed) cases, the event log contains 145,800 event traces. Among these traces, 41 % of the traces end after two events, and 51% of the traces have five or more events. In addition, 62 % of the traces take longer than 100 days to finish. Figure 4.4 depicts the process model mined by an open source SIMPLE algorithm [23].

We first apply Algorithm 1 to extract the time dependent sets for all transitions in the workflow net given in Figure 4.4. The time dependent sets are generated by the tool we developed based on Algorithm 1. Second, using the time dependent sets, we mine the timing constraints from the event log by applying Algorithm 2. The execution results are captured and depicted in Figure 4.5. The process model associated with mined timing constraints is represented as a time Petri net and depicted in Figure 4.6.

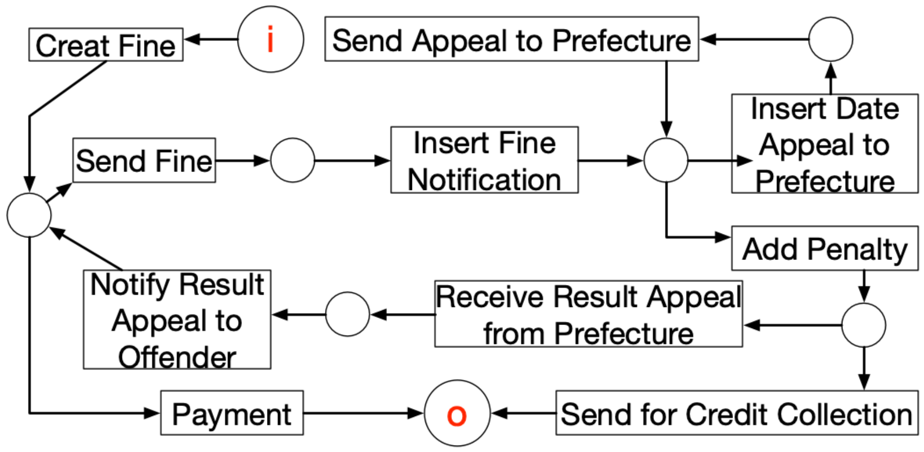
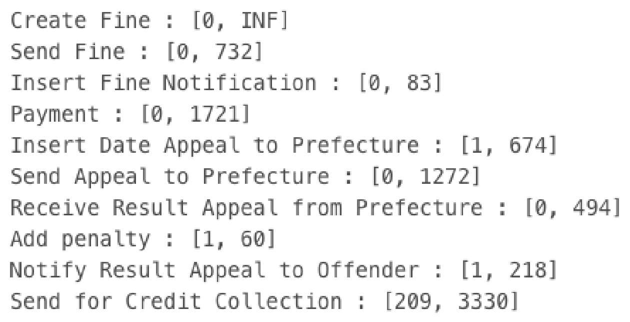

Creat Fine
i
Send Appeal to Prefecture
Send Fine
Insert Fine Notification
Insert Date Appeal to Prefecture
Add Penalty
Notify Result Appeal to Offender
Receive Result Appeal from Prefecture
Payment
o
Send for Credit Collection

Figure 4.4: Road traffic fine management process model

From the mined timing constraints shown in Figure 4.6, we can conclude that except the *Send for Credit Collection* transition, the latest firing times of all other transitions are less than 5 years. The latest firing time of the *Insert Fine Notification* transition is 83 days which obeys Italian law's 90-day notification restrict.

If an offender does not pay the fine on time or if the prefecture denies a offender's appeal, the system sends the offender's information to the credit card company to automatically deduct the fine. According to the law, the fine notification and offender response takes up to 90 and 60 days, respectively. Hence, the *Send for Credit Collection* activity must occur after 150 days. The mined timing constraint for *Send for Credit Collection* activity is $[209, 3330]$ which complies with the law and reflects the actual scenario. The time difference between 150 and 209 is the system processing time for *Send for Credit Collection* activity and/or the appeal process time by prefecture.

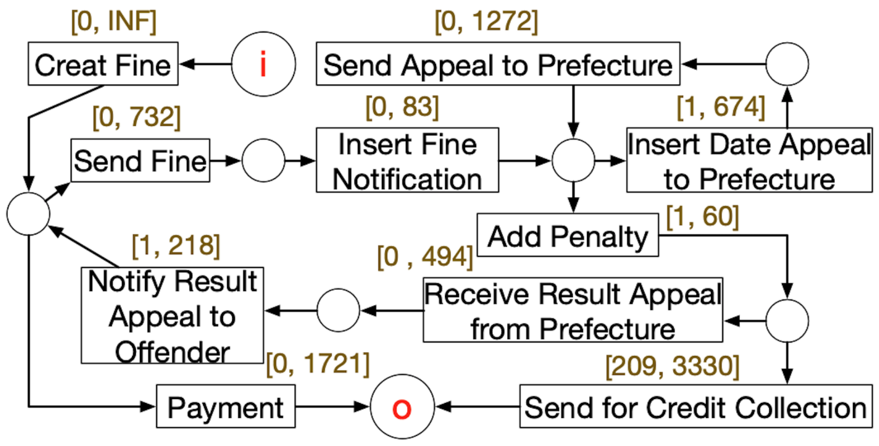
As shown in Figure 4.4, the *Payment* activity has two pre-activities: *Create Fine* and *Notify Result Appeal to Offender*, which represent two scenarios of the *Payment* activity. The first scenario indicated by pre-activity *Create Fine* is that an offender directly pays the fine after the fine creation. The second scenario represented by pre-activity *Notify Result Appeal to Offender* is that an offender appeals against the fine and pays the fine after the appeal is



```
Create Fine : [0, INF]
Send Fine : [0, 732]
Insert Fine Notification : [0, 83]
Payment : [0, 1721]
Insert Date Appeal to Prefecture : [1, 674]
Send Appeal to Prefecture : [0, 1272]
Receive Result Appeal from Prefecture : [0, 494]
Add penalty : [1, 60]
Notify Result Appeal to Offender : [1, 218]
Send for Credit Collection : [209, 3330]
```

Figure 4.5: Timing constraints in traffic fine management process

denied. According to the Italian law, the timing constraint for activity *Payment* under the first scenario is $[0, 60]$, but the law does not specify when the *Payment* must happen if the appeal is denied. However, our algorithm does not distinguish these two scenarios and the mined timing constraint $[0, 1721]$ for the *Payment* activity covers all cases of both scenarios. Hence, it may be loose for the first scenario. Our further work is to extend the proposed approach to tighten the timing constraints.



[0, INF]
Creat Fine
i
[0, 1272]
Send Appeal to Prefecture
[0, 732]
Send Fine
[0, 83]
Insert Fine Notification
[1, 674]
Insert Date Appeal to Prefecture
Add Penalty
[1, 60]
[1, 218]
[0 , 494]
Notify Result Appeal to Offender
Receive Result Appeal from Prefecture
[0, 1721]
[209, 3330]
Payment
o
Send for Credit Collection

Figure 4.6: Process model integrated with timing constraints

4.5 Summary

In this chapter, we have presented an approach that extends existing process mining techniques to not only mine but also integrate timing constraints with a workflow or process model constructed by any existing process mining algorithm. In particular, we first introduced the concept of time dependent set for each transition in a workflow net model and developed an algorithm that automatically finds the sets from a given workflow net model. Based on the time dependent sets, we developed an algorithm to extract implicit timing constraints from event logs for each transition in the model and represent the process model in terms of time Petri net in which we assign each transition with the appropriate timing constraints mined from event logs. The algorithm is implemented based on the open-source process mining framework *pm4py*. A real-life road traffic fine management process scenario is used as a case study to investigate the effectiveness and validity of the approach. The evaluation results show that the algorithm is able to discover the timing constraints of an actual workflow process from event logs.

Chapter 5

Implicit Knowledge Discovery - Process Scenarios

5.1 Background

Due to the complex application design and development, the use of increasingly sophisticated processes, particularly in the utility industry and medical domain, discovering process scenarios has become a subject of great interest. Despite the demonstrated usefulness of process discovery algorithms, these algorithms face challenges in process scenarios discovery [32, 15, 77, 35]. When different scenarios are grouped into one process model not only the accuracy of the model representing the reality reduces, more importantly, the complexity of the model becomes incomprehensible and hence makes it difficult, if not impossible, to achieve the goal of better understanding, monitoring and improving the current processes.

Trace clustering techniques are often used to assist in discovering different process scenarios. Most existing trace clustering methods [15, 34, 61, 16] often contain two major steps. First, use a set of transformation rules to convert each trace in a given event log into a vector.

Second, apply clustering algorithms, such as k-means [61], to the vectors and partition the vectors into different clusters, and hence partition the corresponding event log into different subsets of logs where event traces in the same subset most likely belong to the same scenario. Once an event log is partitioned into different clusters, process discovery techniques are applied to individual clusters to obtain process models for different scenarios.

Event logs often contain rich information, such as activity names, timestamps, activity executors, etc. However, our observation is that most transformation rules used in trace clustering techniques [34, 61, 15, 16] to convert an event trace to a vector only use the activity name information in the event log. We hypothesize that if additional information is added into the constructing vectors, we shall obtain process models that more accurately reflect model fitness and precision criteria. We have also observed that the commonly used trace clustering algorithms , e.g., k-means [61, 16, 15], need a priori knowledge about the number of scenarios k, which is unfortunately not available for most application. The question is, do we have an approach to offer a good solution without the requirements of domain knowledge related to the number of scenarios? The last observation is that the shape of all clusters found by commonly used trace clustering techniques [61, 16, 15] is convex, which is very restrictive. Hence, we should propose an approach for discovering clusters with arbitrary shapes.

In this chapter, we present an approach that uses timing information to assist in discovering process scenarios from event logs. This approach has three steps. First, we obtain the time dependent sets from a process model produced by applying an existing process discovery algorithm to the entire event logs. Second, we construct aggregated vectors containing activity and timing information derived from the time dependent set and event time stamps. For the third step, we provide two algorithms, the distance-based and the density-based algorithms, to generate subsets of event logs where event traces in the same subset are most likely under the same scenarios. A real wastewater treatment process provided by a domain expert is used as a case study to investigate the effectiveness and validity of the approach.

To evaluate the performance of the proposed approach for process scenario discoveries, we apply an existing process discovery algorithm, e.g., the *HeuristicsMiner* algorithm [81], to obtain a process model from each subset.

5.2 Related Work

The work in [31] is the first study which applies trace clustering techniques to assist in discovering the process scenarios in order to obtain more accurate and interpretable process models from event logs. This study proposed an approach to cluster the event traces based on the structural patterns frequently occurring in the event log, and then partition the corresponding event log into different subsets of logs where event traces in the same subset most likely belong to the same scenario. Once the event logs are partitioned into different clusters, process discovery techniques can be applied on the clusters to obtain process models for different scenarios.

The most straightforward idea for clustering traces methods relies on traditional data clustering techniques [40]. Greco et al. [34] were pioneers in studying the clustering of log traces within the process mining domain. They use a vector space model considering the activities to cluster the traces in an event log with the purpose of discovering more simple process models for the subgroups. The authors propose the use of disjunctive workflow schemas (DWS) for discovering process models. The underlying clustering methodology is k-means clustering. Song et al. [61] proposed an approach to construct a vector space model for traces in an event log. In [61], the author allows for different kinds of attributes, e.g., activity name, executor, to determine the vector associated with each event trace, and then provide four clustering techniques for trace clustering. In [15, 16], they extend the existing trace clustering techniques by improving the way in which control-flow context information is taken into account. The control-flow context information refers to the execution pattern of activities in the event log. Jagadeesh et al. [15] propose a generic edit distance technique based on the Levenshtein distance. The approach relies on substitution, insertion, and deletion costs to partition the event log into a set of sub logs. Bose et al. [16] propose a refinement of the technique by using conserved patterns, which are *Maximal*, *Super Maximal*, and *Near Super Maximal Repeats*. However, their implementation applies hierarchical clustering instead of k-means.

5.3 Construct Aggregated Vectors with Activity and Timing Information

In this Chapter, we present an approach to automatically construct aggregated vectors that contain both activity name from event traces and timing information derived from the time dependent set and event entry timestamps.

We first apply a similar approach used in [61] to map each event trace σ in a given event log into an *activity vector* ($\vec{\mathcal{A}}_\sigma$) which is defined below.

Definition 17 (Activity Vector ($\vec{\mathcal{A}}$))**.** *Given an event trace σ and an activity set $\mathit{\Omega}$, the activity vector $\vec{\mathcal{A}}$ corresponding to an event trace σ is $\vec{\mathcal{A}}_\sigma = (\mathcal{N}_1(\alpha_1), \cdots, \mathcal{N}_n(\alpha_i))$, where $n = |\mathit{\Omega}|, \alpha_i \in \mathit{\Omega}$, and $\mathcal{N}(\alpha_i)$ is the number of times the activity α_i occurs in the trace σ.*

Based on the definition, for event log log given in Table 3.1, we have the activity set $\mathit{\Omega} = \{A, B, C, D, E\}$, hence the size of all activity vector $|\vec{\mathcal{A}}| = 5$. Furthermore, for **Trace 1** (σ_1), its corresponding activity vector $\vec{\mathcal{A}}_{\sigma_1} = (1, 1, 1, 1, 0)$, where $\vec{\mathcal{A}}_{\sigma_1}[5] = 0$ indicates that activity E does not occur in **Trace 1**.

There is another important information contained in an event trace, i.e., the time stamp of each recorded activity. Our idea is to extend the activity vector by adding timing information related activities in a given trace. Our hypothesis is with additional activity related to timing information, we will be able to discover more accurate scenarios. Our experiments confirms this hypothesis. To obtain the timing information related to each activity in a given trace, we first introduce the following definitions.

Definition 18 (Time Dependent Pair (α, β))**.** *Given an event log's activity set Ω, for any two activities $\alpha, \beta \in \Omega$, if $\beta \in \Theta(\alpha)$, the activities α and β are called a time dependent pair denoted as (α, β), and the set of all time dependent pairs in a given activity set Ω is denoted as $\mathcal{O} = \bigcup_{\alpha \in \Omega} \{(\alpha, \beta) | \beta \in \Theta(\alpha)\}$.*

For instance, in Example 2, activity D's time dependent set is $\Theta(D) = \{B, C, E\}$, the corresponding time dependent pairs are $(D, B), (D, C)$, and (D, E). Furthermore, the set of all time dependent pairs corresponding to the event log shown in Table 3.1 is $\mathcal{O} = \{(B, A), (C, A), (D, B), (D, C), (D, E), (E, A)\}$.

Definition 19 (Timing Vector $(\vec{\mathcal{T}})$)**.** *Given a time dependent pair set $\mathcal{O}$, the timing vector $\vec{\mathcal{T}}$, $|\vec{\mathcal{T}}| = |\mathcal{O}|$, represents the maximal time durations between two activities of a given time dependent pair $(\alpha, \beta) \in \mathcal{O}$. For a given trace $\sigma = [e_1, \cdots, e_m]$, the value of the timing vector $\vec{\mathcal{T}_\sigma}[(\alpha, \beta)]$ is defined as: $\vec{\mathcal{T}_\sigma}[(\alpha, \beta)] = \max\{\mathcal{F}_\tau(e_i) - \mathcal{F}_\tau(e_j) \mid 1 \le i, j \le m \land i \ne j \land e_i, e_j \in \sigma \land \mathcal{F}_\alpha(e_i) = \alpha \land \mathcal{F}_\alpha(e_j) = \beta\}$.*

Consider **Trace** $1(\sigma_1)$ in Table 3.1 and a time dependent pair (B, A). The trace σ_1 contains the activity B at time "10:24", the activity A's time stamp is "08:15". The time duration between the activity A's time stamp and the activity B's time stamp is 129 with minutes as the time units. Hence we have $\vec{\mathcal{T}_{\sigma_1}}[(B, A)] = 129$, indicating that activity B occurs at most 129 time units after the occurrence of activity A in σ_1. Algorithm 3 gives the detailed steps of obtaining the timing vector from an event trace.

Before we aggregate the activity and timing vectors to obtain a vector that contains both activity information and timing information in an event trace, we normalize the activity and

Algorithm 3 CONSTRUCTING TIMING VECTOR

```
Require: An event trace $\sigma = \{e_1, \cdots, e_n\}$ and the set of all time dependent pairs $\mathcal{O} = \bigcup_{\alpha \in \Omega} \{(\alpha, \beta) | \beta \in \Theta(\alpha)\}$
Ensure: The corresponding timing vector $\vec{\mathcal{T}_\sigma} = (\Gamma(\alpha_1, \beta_1), \cdots, \Gamma(\alpha_i, \beta_j))$, where the $\Gamma(\alpha_i, \beta_j)$ indicates the maximal time duration
    between two activities of a given time dependent pair $(\alpha_i, \beta_j) \in \mathcal{O}$
 1: for $i = 1$ to $n$ do
 2:     $j = i - 1$
 3:     while $j \neq 0$ do
 4:         if $(e_i, e_j) \in \mathcal{O}$ then
 5:             if $\mathcal{F}_\tau(e_i) - \mathcal{F}_\tau(e_j) > \Gamma(\mathcal{F}_\alpha(e_i), \mathcal{F}_\alpha(e_j))$ then
 6:                 $\Gamma(\mathcal{F}_\alpha(e_i), \mathcal{F}_\alpha(e_j)) = \mathcal{F}_\tau(e_i) - \mathcal{F}_\tau(e_j)$
 7:             end if
 8:         end if
 9:         $j = j - 1$
10:     end while
11: end for
```

timing vectors. Normalization of the activity and timing vector is used to prevent features with large numerical values from dominating in distance-based objective functions which is used in trace clustering. For a given event trace σ, the corresponding activity vector and timing vector are $\vec{\mathcal{A}} = (a_1, \cdots, a_m)$ and $\vec{\mathcal{T}} = (t_1, \cdots, t_n)$, respectively, we apply $\frac{\vec{\mathcal{A}}}{\|\vec{\mathcal{A}}\|}$ and $\frac{\vec{\mathcal{T}}}{\|\vec{\mathcal{T}}\|}$ to normalize the vectors, where $\|\vec{\mathcal{A}}\| = \sqrt{a_1^2 + \cdots + a_n^2}$ and $\|\vec{\mathcal{T}}\| = \sqrt{t_1^2 + \cdots + t_n^2}$. Finally, we aggregate the normalized activity and normalized timing vectors.

Definition 20 (Aggregated Vector ($\vec{\mathcal{V}}$))**.** *Given an event trace σ, assume the corresponding activity and timing vectors are $\vec{\mathcal{A}_\sigma} = (a_1, \cdots, a_i, \cdots, a_m)$ and $\vec{\mathcal{T}_\sigma} = (t_1, \cdots, t_j, \cdots, t_n)$, respectively, the aggregated vector $\vec{\mathcal{V}}$ corresponding to an event trace σ is $\vec{\mathcal{V}_\sigma} = (\frac{a_1}{\|\vec{\mathcal{A}}\|}, \cdots, \frac{a_i}{\|\vec{\mathcal{A}}\|}, \cdots, \frac{a_m}{\|\vec{\mathcal{A}}\|}, \frac{t_1}{\|\vec{\mathcal{T}}\|}, \cdots, \frac{t_j}{\|\vec{\mathcal{T}}\|}, \cdots, \frac{t_n}{\|\vec{\mathcal{T}}\|})$, where $a_i = \vec{\mathcal{A}_\sigma}[i]$, $t_j = \vec{\mathcal{T}_\sigma}[j]$, $\|\vec{\mathcal{A}}\| = \sqrt{a_1^2 + \cdots + a_n^2}$, and $\|\vec{\mathcal{T}}\| = \sqrt{t_1^2 + \cdots + t_n^2}$.*

Consider **Trace** $1(\sigma_1)$ in Table 3.1, the aggregated vector corresponding to σ_1 is $\vec{\mathcal{V}_{\sigma_1}} = (0.5, 0.5, 0.5, 0.5, 0, 0.419, 0.423, 0.569, 0.566, 0, 0)$. Table 5.1 gives the vector value for the other four traces. Note that, we can also use weighted concatenation to aggregate the normalized activity and timing vectors. The weighted values provide a way to emphasize

	Activity Information					Timing Information					
	$\mathcal{N}(A)$	$\mathcal{N}(B)$	$\mathcal{N}(C)$	$\mathcal{N}(D)$	$\mathcal{N}(E)$	(B,A)	(C,A)	(D,B)	(D,C)	(D,E)	(E,A)
$\vec{\mathcal{V}}_{\sigma_1}$	0.500	0.500	0.500	0.500	0.000	0.419	0.423	0.569	0.566	0.000	0.000
$\vec{\mathcal{V}}_{\sigma_2}$	0.500	0.500	0.500	0.500	0.000	0.759	0.458	0.004	0.439	0.000	0.000
$\vec{\mathcal{V}}_{\sigma_3}$	0.500	0.500	0.500	0.500	0.000	0.144	0.742	0.652	0.053	0.000	0.000
$\vec{\mathcal{V}}_{\sigma_4}$	0.500	0.500	0.500	0.500	0.000	0.624	0.322	0.305	0.642	0.000	0.000
$\vec{\mathcal{V}}_{\sigma_5}$	0.577	0.000	0.000	0.577	0.577	0.000	0.000	0.000	0.000	0.151	0.988

Table 5.1: Aggregated vectors transferred from event logs

which perspective should have more impact on the outcome of discovery process scenarios.

5.4 Process Scenarios Discovery

In this Chapter, we provide two algorithms as an option to generate subsets of event logs where event traces in the same subset are most likely under the same scenarios. The first algorithm is the distance-based algorithm named the two-phase approach for discovering the process scenarios from a given event log using aggregated vectors. Another algorithm is the density-based process discovery algorithm based on aggregated vectors. In the following, we will explain in detail.

5.4.1 Distance-based Process Discovery Algorithm

For a given event log, often times, we do not know aprior how many different scenarios are embedded in the log. Hence, the first phase of our two-phase approach is to identify the number of possible scenarios k in a given log. The second phase is to apply an existing clustering approach, such as the k-means algorithm [61], to partition the event log into k clusters based on the aggregated vectors so that existing process discovery algorithms can be applied on the clusters of sub-logs to obtain different process scenarios in terms of process models.

We use Figure 5.2 to illustrate the basic idea used in the first phase to obtain the number of possible scenarios k. For simplicity, assume the cardinality of event trace's aggregated vector $|\vec{\mathcal{V}}| = 2$, therefore, each aggregated vector (a trace) can be mapped to a point in a two-dimensional domain shown in Figure 5.2. Consider the centroid point A, we can define two radius circles, the inner circle, and the outer circle. The points within the inner circle of A have short distance from A, or can be considered 'similar' to A, while the points outside the outer circle (such as C) are considered far away or different from A. The radius are the thresholds for defining if two points are similar or different. The points outside the inner circle but within the outer cycle, such as point B, may or may not have similar characteristics to A. For these points, we repeat the process of choosing a centroid point and defining similar and different points from the new centroid points, until all points are categorized. Then the number of centroids selected during the process is the k. There are many existing techniques in the literature to identify the radius of inner and outer circles, i.e., the distance thresholds, such as the cross validation approach [17]. Sometimes, the distance threshold can also be set by domain experts [34]. Here, we adopt the upper quartile and the lower quartile as the two distance thresholds.

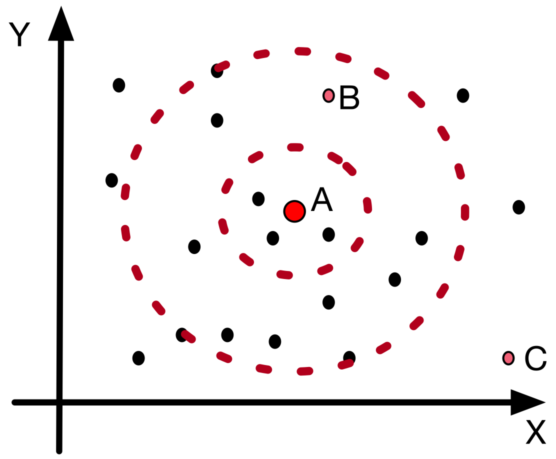


Figure 5.1: Basic idea used in distance-based approach for obtaining the number of possible scenarios k

Let $d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j)$ denote *Euclidean* distances between aggregated vectors $\vec{\mathcal{V}}_i$, $\vec{\mathcal{V}}_j$, Algorithm 5 shows the procedure to find the number of the scenarios from aggregated vectors. The time complexity of the algorithm is $O(N^2)$, where N is the number of event traces in the event log log. We use Example 5 to illustrate Algorithm 5.

Algorithm 4 OBTAINING THE NUMBER OF POSSIBLE SCENARIOS k IN EVENT LOGS

```
Require: A set of aggregated vectors $\Pi = \{\vec{\mathcal{V}}_1, \cdots, \vec{\mathcal{V}}_{|\log|}\}$ and a set of Euclidean distances
    $\mathcal{M} = \{d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j) | \vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j \in \Pi\}$; $\delta_{min}$ and $\delta_{max}$ be the two distance thresholds, respectively.
Ensure: The number of clusters ($k$) and a set of centroids $\mathcal{C}$ for the $k$ clusters' aggregated
    vectors
 1: $k \leftarrow 0$ and $\mathcal{C} \leftarrow \emptyset$
 2: while $\Pi \neq \emptyset$ do
 3:     Select an arbitrary aggregated vector $\vec{\mathcal{V}}_i \in \Pi$
 4:     $\Pi \leftarrow \Pi - \{\vec{\mathcal{V}}_i\}$
 5:     $\Delta \leftarrow \{\vec{\mathcal{V}}_i\}$
 6:     $k \leftarrow k + 1$
 7:     for each $\vec{\mathcal{V}}_j \in \Pi$ do
 8:         if $d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j) \leq \delta_{max}$ then
 9:             $\Delta \leftarrow \Delta \cup \{\vec{\mathcal{V}}_j\}$
10:         end if
11:         if $d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j) \leq \delta_{min}$ then
12:             $\Pi \leftarrow \Pi - \{\vec{\mathcal{V}}_j\}$
13:         end if
14:     end for
15:     $\vec{c} \leftarrow$ calculate the initial centroid of the aggregated vectors in the set $\Delta$
16:     $\mathcal{C} \cup \{\vec{c}\}$
17: end while
18: return $k$ and $\mathcal{C}$
```

Example 4. *Consider the set of aggregated vectors* $\Pi = \{\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4, \vec{\mathcal{V}}_5\}$ *given in the Table 5.1, the set of Euclidean distances of two aggregated vectors in L is* $\mathcal{M} = \{(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2) = 166.39, (\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_3) = 237.58, (\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_4) = 108.03, (\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_5) = 343.18, (\vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3) = 315,$
$(\vec{\mathcal{V}}_2, \vec{\mathcal{V}}_4) = 91.28, (\vec{\mathcal{V}}_2, \vec{\mathcal{V}}_5) = 306.72, (\vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4) = 299.03, (\vec{\mathcal{V}}_3, \vec{\mathcal{V}}_5) = 405.6, (\vec{\mathcal{V}}_4, \vec{\mathcal{V}}_5) = 323.32\}$, *and the first quartile and the third quartile of the elements in* $\mathcal{M}$ *is* $\delta_{min} = 5$ *and* $\delta_{max} = 10$, *respectively.*

According to Line 4-6 of algorithm, we select an aggregated vector $\vec{\mathcal{V}}_1$*, remove it from the set* Π*, and put it into the set* Δ*. Based on the line 7, the number of the scenario becomes one, i.e.* $k = 1$*. We apply Line 8-15 of algorithm for every aggregated vector in the the set* Π*. For the* *aggregated vector* $\vec{\mathcal{V}}_2$*, since the distance between aggregated vectors* $\vec{\mathcal{V}}_1$ *and* $\vec{\mathcal{V}}_2$*, i.e.* $d(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2)$*, is* 166.39 *and less than* δ_{min}*, we remove the aggregated vector* $\vec{\mathcal{V}}_2$ *from the set* Π *and add* $\vec{\mathcal{V}}_2$ *into the set* Δ*. Hence, the set* Δ *is updated to* $\{\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2\}$*, and* Π *becomes* $\{\vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4, \vec{\mathcal{V}}_5\}$*. For the aggregated vector* $\vec{\mathcal{V}}_3$*, the distance between aggregated vectors* $\vec{\mathcal{V}}_1$ *and* $\vec{\mathcal{V}}_3$*, i.e.* $d(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_3)$*, is* 237.58 *and less than* δ_{max}*, we add* $\vec{\mathcal{V}}_3$ *into set* Δ*. Because* $d(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_3) > \delta_{min}$*, we keep* $\vec{\mathcal{V}}_3$ *in the set* Π*. After applying the procedure to the other two aggregated vectors, the* $\Delta = \{\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4\}$*, and the* $\Pi = \{\vec{\mathcal{V}}_3, \vec{\mathcal{V}}_5\}$*. We have the initial centroid of the aggregated vectors in the set* Δ *is* $(1, 1, 1, 1, 0, -790, -87, -327, 0, 0, 0)$*. We repeat the steps until* Π *becomes empty, and have* $k = 3$ *and the initial centroids are* $(1, 1, 1, 1, 0, -790, -87, -327, 0, 0, 0)$*,* $(1, 1, 1, 1, 0, 54, 279, 245, 20, 0, 0)$*, and* $(1, 0, 0, 1, 1, 0, 0, 0, 0, 23, 151)$*.*

In the second phase, we take the number of scenarios k and the initial centroids produced by the first phase as input and apply k-means algorithm [43] to partition the event log into k clusters, i.e., scenarios. Finally, the existing process discovery algorithms can be applied on the subset of logs to obtain different process scenarios in terms of process models.

5.4.2 Density-based Process Discovery Algorithm

In Chapter 5.4.1, we propose a distance-based algorithm to generate subsets of event logs where event traces in the same subset are most likely under the same scenarios. The distance-based algorithm is easy to understand and implement and can handle large datasets well. However, the distance-based algorithm gets difficult in high dimensional spaces as the distance between the points increases and Euclidean distance diverges. More importantly, clusters can take any irregular shape unlike the distance-based algorithm where clusters are more or less spherical due to the characteristics of a process such as wastewater treatment processes. Hence, we proposed the density-based algorithm for process scenario discovery.

We use Figure 5.2 to illustrate the basic idea used in the density-based algorithm to partition the event log into a set of clusters. For simplicity, assume the cardinality of event trace's aggregated vector $|\vec{\mathcal{V}}| = 2$, therefore, each aggregated vector (an event trace) can be mapped to a point in a two-dimensional domain shown in Figure 5.2.

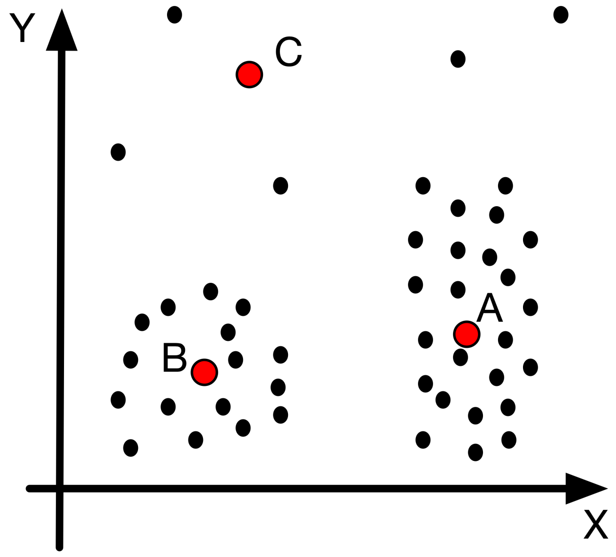


Figure 5.2: Strategy used in the density-based algorithm to partition event logs into a set of clusters

Considering the sample sets of aggregated vectors depicted in Figure 5.2, we can easily and clearly detect two different process scenarios of aggregated vectors and noise not belonging to any of those scenarios. For instance, aggregated vectors A and B belong to two different process scenarios. However, the aggregated vector C is noise. The main reason we recognize the clusters is that within each cluster, we have a typical density of points that is considerably higher than outside of the cluster. Furthermore, the density within the areas of noise is lower than the density in any of the clusters. The key idea is that for each point of a cluster the neighborhood of a given radius has to contain at least a minimum number of points, which the density in the neighborhood has to exceed a threshold. Hence, we would have to define the two appropriate parameters which are the following. The distance that will be used to locate the points in the neighborhood of any point and the minimum number of points clustered together for a region to be considered dense. Then, we can retrieve all points that are reachable from the given point using the these two parameters. There is no easy way to get the knowledge related to the parameters in advance, the existing technique in the literature to determine the parameters of the cluster is heuristic [28, 48, 79]. Sometimes, the two parameters can be set by domain experts [34].

Let $d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j)$ denote *Euclidean* distances between aggregated vectors $\vec{\mathcal{V}}_i$, $\vec{\mathcal{V}}_j$. Let *Eps* denote the distance measure that will be used to locate the points in the neighborhood of any point. Let $MinPts$ denote the minimum number of points clustered together for a region to be considered dense.

Algorithm 5 shows the procedure to partition the event log into a set of clusters using the density-based algorithm from aggregated vectors. The time complexity of the algorithm is $O(n \times log(n))$, where n is the number of event traces in the event log log. We use Example 5 to illustrate Algorithm 5.

Example 5. *Consider the set of aggregated vectors* $\Pi = \{\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4, \vec{\mathcal{V}}_5\}$ *given in the Table 5.1, the set of Euclidean distances of two aggregated vectors in L is* $\mathcal{M} = \{(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2) = 0.203, (\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_3) = 0.202, (\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_4) = 0.108, (\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_5) = 0.508, (\vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3) = 0.306, (\vec{\mathcal{V}}_2, \vec{\mathcal{V}}_4) = 0.124$, $(\vec{\mathcal{V}}_2, \vec{\mathcal{V}}_5) = 0.507, (\vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4) = 0.281, (\vec{\mathcal{V}}_3, \vec{\mathcal{V}}_5) = 0.508, (\vec{\mathcal{V}}_4, \vec{\mathcal{V}}_5) = 0.508\}$*, and the two parameters are* $Eps = 0.3$ *and* $MinPts = 1$.

According to Lines 2-5, we select an aggregated vector $\vec{\mathcal{V}}_1$*, and check the label of aggregated vector* $\vec{\mathcal{V}}_1$*. We apply Line 6-11 of algorithm for every aggregated vector in the set* Π *to find the neighborhood of aggregated vector* $\vec{\mathcal{V}}_1$*. Since the distance between aggregated vectors* $\vec{\mathcal{V}}_1$ *and* $\vec{\mathcal{V}}_2$*, i.e.* $d(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2)$*, is* 0.203 *and less than Eps, we add* $\vec{\mathcal{V}}_2$ *into list* $NeighborsList$*. After applying the step to the other three aggregated vectors* $\vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4, \vec{\mathcal{V}}_5$*, the* $NeighborsList$ *becomes* $\vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4$*. Then we check whether the list* $NeighborsList$ *contains at least a* $MinPts$ *of points on Line 12-15. Based on Line 16 and 17, aggregated vector* $\vec{\mathcal{V}}_1$ *is labeled as 1 and then removed from list* $NeighborsList$*. We apply Line 18-32 of algorithm to find the neighborhood of every aggregated vector in the list* $NeighborsList$*, i.e.*$\vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4$*. We repeat the steps until*

Algorithm 5 Event log partitioning

```
Require: A set of aggregated vectors $\Pi = \{\vec{\mathcal{V}}_1, \cdots, \vec{\mathcal{V}}_{|\log|}\}$ and a set of Euclidean distances
    $\mathcal{M} = \{d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j) | \vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j \in \Pi\}$; Eps and MinPts be the two parameters.
Ensure: a set of the clusters related to aggregated vectors
 1: $k \leftarrow 0$
 2: for each $\vec{\mathcal{V}}_i \in \Pi$ do
 3:   if $\vec{\mathcal{V}}_i$ is not labeled then
 4:     continue
 5:   end if
 6:   $NeighborsList \leftarrow$ Empty List
 7:   for each $\vec{\mathcal{V}}_j \in \Pi$ do
 8:     if $d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j) \leq Eps$ then
 9:       $NeighborsList \leftarrow NeighborsList \cup \{\vec{\mathcal{V}}_j\}$
10:     end if
11:   end for
12:   if $|NeighborsList| < MinPts$ then
13:     $\vec{\mathcal{V}}_i$ is labeled as noise
14:     continue
15:   end if
16:   $k \leftarrow k + 1$ and $\vec{\mathcal{V}}_i$ is labeled as $k$
17:   $SearchList \leftarrow NeighborsList \backslash \{\vec{\mathcal{V}}_i\}$
18:   for each $\vec{\mathcal{V}}_m \in SearchList$ do
19:     if $\vec{\mathcal{V}}_m$ is labeled as noise then
20:       $\vec{\mathcal{V}}_m$ is labeled as $k$
21:     end if
22:     $\vec{\mathcal{V}}_m$ is labeled as $k$
23:     $NeighborsList \leftarrow$ Empty List
24:     for each $\vec{\mathcal{V}}_n \in \Pi$ do
25:       if $d(\vec{\mathcal{V}}_m, \vec{\mathcal{V}}_n) \leq Eps$ then
26:         $NeighborsList \leftarrow NeighborsList \cup \{\vec{\mathcal{V}}_n\}$
27:       end if
28:     end for
29:     if $|NeighborsList| \geq MinPts$ then
30:       $SearchList \leftarrow SearchList \cup NeighborsList$
31:     end if
32:   end for
33: end for
```

all aggregated vectors are labeled, and partition the event log into two subsets, i.e., scenarios. The first subset of the event log contain $\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4$, and the second subset contain $\vec{\mathcal{V}}_5$

Finally, the existing process discovery algorithms can be applied on the subset of logs to obtain different process scenarios in terms of process models.

5.5 Experimental Evaluation

In this Chapter, we firstly apply the technique to the artificial logs to investigate the effectiveness and validity. Then we experimentally evaluate the proposed approach on five real-life event logs [1, 2, 3, 5, 4].The characteristics of these event logs are summarized in Table 5.2. The artificial logs are generated from the Processes and Logs Generator (**Plg**) [18], which is an open-source tool that is used to generate artificial event logs based on the user-defined process model.

The objectives of our evaluations consist of four components. First, investigate the effectiveness of process scenario discoveries in wastewater treatment. To do so, we apply the proposed approach on the artificial logs generated from the **Plg** based real wastewater treatment process provided by a domain expert and examine the effectiveness of the approach. Second, we evaluate if having timing information can improve the quality of process scenario models discovered from event logs. In order to achieve the objective, we choose an existing heuristic mining algorithm [81], and apply it to the same event logs but with and without time information. Third, we compare the number of scenarios and the scenario models obtained by the distance-based algorithm with k-means approach where k is exhaustively searched. The last, we compare the density-based algorithm with the four commonly used process scenarios discovery approaches such as the *k-means* approach (**KM**) [61], the *Model-based* approach (**MB**) [25], the *Distanced-based* approach (**DB**) [34], and the *Conserved Patterns* approach (**CP**) [16] using real-life event logs. Before the evaluation, we first define our evaluation criteria.

DataSet	Number of Traces	Number of Event Entry	Average number of Event Entry Per Trace
BPIC2013 [1]	819	2351	2.871
BPIC2020 [2]	10500	56,437	5.374
Hospital Billing [3]	100000	451359	4.514
Review Process [5]	10000	236360	23.636
Road Traffic [4]	150370	561470	3.734

Table 5.2: Characteristics of event logs used for evaluations in the discovery phase

5.5.1 Performance Metrics

The quality of a process model is evaluated by two criteria, i.e., (1) fitness f, and (2) precision p. *Fitness* measures how well a model can reproduce the process behavior contained in a log, and the *precision* measures the degree to which the behavior made possible by a model is found in a log [20].The higher fitness value and precision value, the better quality of the process model. We use the pm4py [8] library to calculate the fitness value and precision value.

As we may have multiple scenarios in a given event log, we need the weighted average fitness f^W and precision p^W to evaluate the quality of the multiple scenario process models. Similar to the approach in [25], the weighted average fitness and precision are calculated as follows:

$$f^W = \frac{\sum_{i=1}^{k} n_i \times f_i}{|\log|}$$
$$p^W = \frac{\sum_{i=1}^{k} n_i \times p_i}{|\log|}$$

where k is the number of subsets of logs representing the scenarios, n_i is the number of event traces in the ith subset of a given event log, and f_i and p_i are the fitness value and precision value of the process model derived from of subsets of logs, respectively.

The fitness and precision are two aspects of a process model which may not always be consistent. The $F1$ score [69] is defined as the harmonic mean of the weighted average fitness f^W and precision p^W, i.e. $F1 = \frac{2 \times f^W \times p^W}{f^W + p^W}$. We will also use the $F1$ score as an evaluation criteria.

5.5.2 Effectiveness Validation

This set of experiments examines the proposed approach's effectiveness for process scenario discovery in the wastewater treatment domain. More specifically, based on the process model provided by the domain expert, we first use the **Plg** to generate five artificial event logs by injecting an incremental amount of noise ranging from 1% to 20% to produce an event log that contains one thousand event traces. Then, for each artificial event log, we apply the proposed approach to partition the event log into clusters and discover a process model from each subset of the event log using the heuristic mining algorithm presented in [81].

Figure 5.3 shows the wastewater treatment process model in the form of the Business Process Modeling Notation [85]. All wastewater treatment processes begin with influent wastewater from sewage lines arriving at a treatment facility. The wastewater is sent through coarse and fine screens to remove both large and small objects, debris, etc. from the water. After these first mandatory steps are completed, the wastewater will go through different sub-processes depending on if the water can be reused or if it will be disposed of at a landfill. Figure 6.3, Figure 6.4, and Figure 6.5 show the results produced by the proposed approach.

The first scenario within the wastewater treatment process shown in Figure 6.3 is used to remove solid debris in the water that ranges anywhere from 0.01 inch to 6 inches. This process begins at the "Coarse Screens" activity, where the screens have openings of 0.25 inches to 6 inches. These screens remove the larger solids, such as rags, sticks, rocks, etc. After the large debris is removed from the wastewater, it moves to the "Fine Screens" activity where smaller particles are moved from the water. Screens anywhere from 0.01 inch to 0.25 inches are used to suspend solids and clarify the water to acceptable levels. The final processing step is the

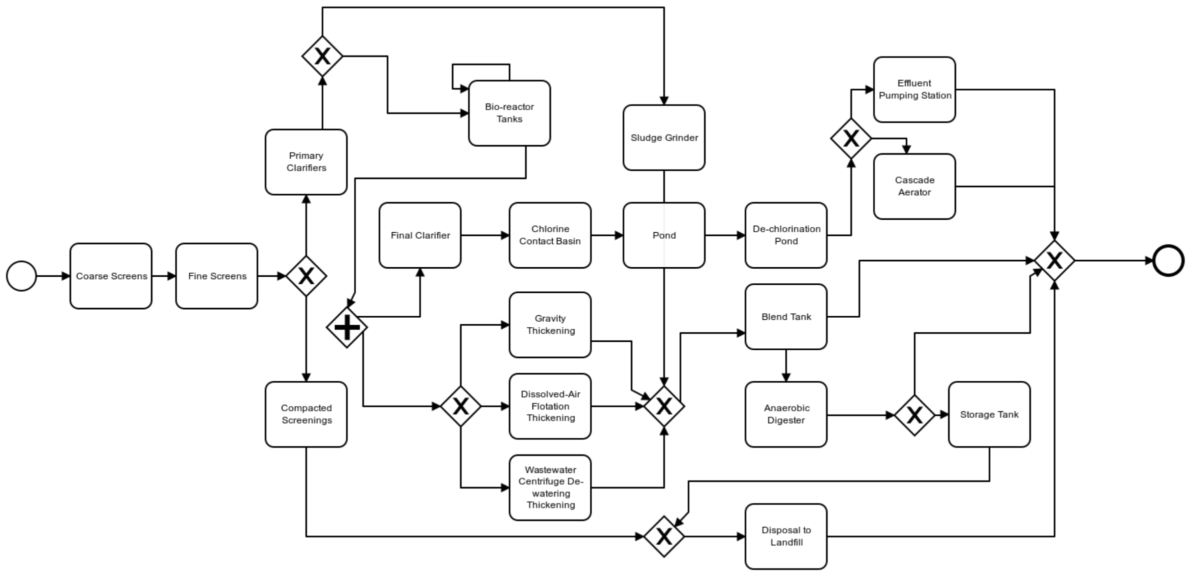


Figure 5.3: Wastewater Treatment Process Model

“Compacted Screenings” where after the screens used previously are washed, and they are then compacted to reduce the amount of water and increase the concentration of solids. The first scenario ends with the “Disposal to Landfill” activity, where waste is disposed of.

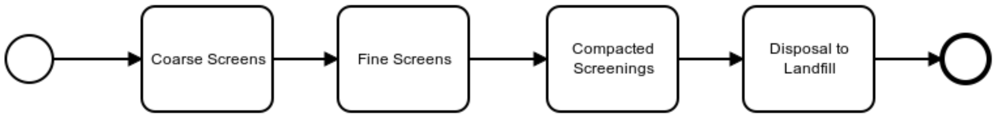


Figure 5.4: Scenario one: Removing solid debris process

The second scenario within the wastewater treatment process shown in Figure 6.4 is used to treat sludge by converting it to a solid form. The same process of water going through the coarse/fine screens occurs as in the first scenario. Then, the water goes through the “Primary Clarifiers” activity, where solids are separated from the wastewater by gravity in a clarification tank. After this activity, “Blending” routes some of the water flow away from the treatment process so that it can be blended with fully treated water before it is disposed of. The “Anaerobic Digester” step continues the treatment process by degrading organic matter by removing oxygen. The process can either end at this point or continue with the “Storage Tank” activity where water can be stored before it is disposed of at a landfill.

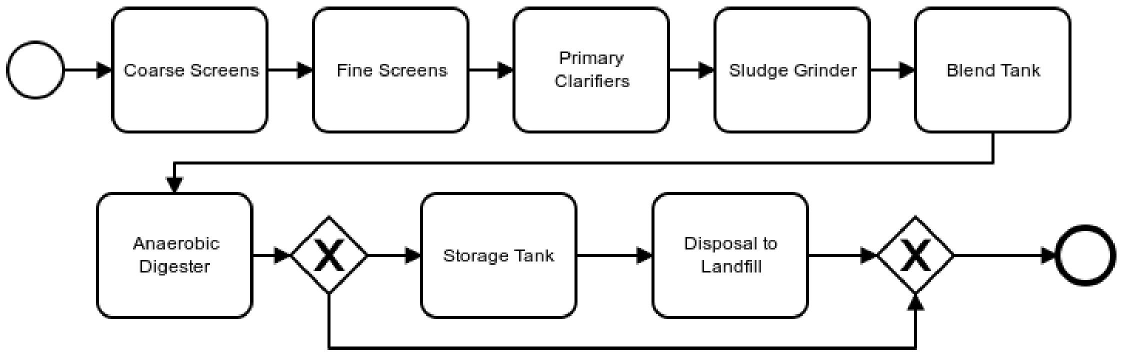


Figure 5.5: Scenario two: Treating sludge process

The third scenario within the wastewater treatment process shown in Figure 6.5 describes the liquid flow through the process. The third scenario also starts with coarse/fine screen activities. It then goes to the "Primary Clarifiers" activity, similar to the second scenario, to allow solids to separate from the liquids. The liquid then moves to the "Bio-reactor Tanks," where the liquid is treated by microfiltration membrane bioreactors that combine a suspended growth biological reactor with microfiltration for secondary water treatment. Then, depending on various chemical levels, clarity levels, etc., the water will simultaneously travel to various other activities before the end of the process. The first route is to the "Final Clarifier" activity, where an activated sludge process is used to settle out microorganisms. Then the "Chlorine Contact Basin" is also used to disinfect the water and remove microorganisms, bacteria, viruses, etc., from the water. Water is then moved to "Ponds," where organic content and pathogens are reduced/removed from the wastewater. Then the "De-chlorination Pond" reduces chlorine levels in the wastewater. Before the process ends, the water is moved to either the "Effluent Pumping Station" for transporting the water or the Cascade Aerator to oxidize iron and reduce dissolved gases. The third scenario's second route starts with one of three activities. The first, "Gravity Thickening," is used to condense biosolids to separate the solids from the solid-free supernatant. The second activity, "Dissolved-Air Flotation Thickening", thickens sludge to bring solids to the surface, separating the solids and liquids. The final option is "Wastewater Centrifuge De-watering Thickening," where a high-speed rotation separates water from the sewage. After one of these processes is used to treat the wastewater, it is then sent to the "Blend Tank" and the "Anaerobic Digester" for further treatment before the process ends.

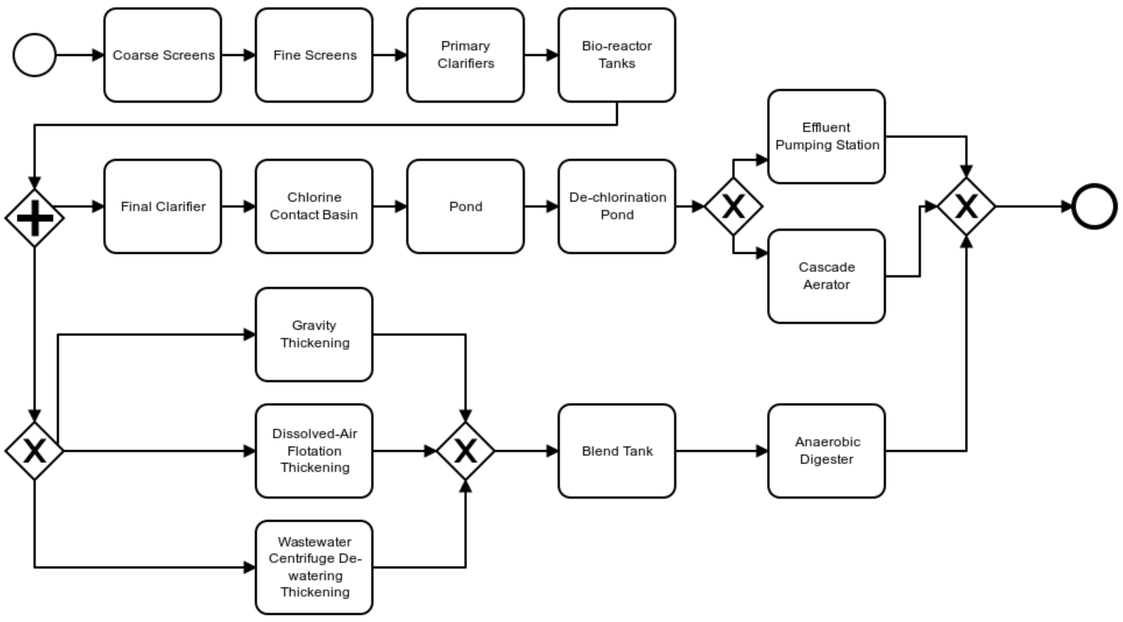


Figure 5.6: Scenario three: Treating liquid process

According to the wastewater treatment workflow from conventional water reclamation plants, wastewater treatment is a process in which the solids in wastewater are partially removed and partially changed by decomposition from highly complex, putrescible organic solids to mineral or relatively stable organic solids. Primary and secondary treatment removes the majority of biological oxygen demand and suspended solids in wastewater. The proposed approach utilizes the duration of each treatment activity and the frequency of event trace sequences related to treatment processes for different waste forms to discover the process scenarios in wastewater treatment. Based on the domain expert's verification, the evaluation results show that the proposed approach is able to discover the process scenarios from event logs in the wastewater treatment domain.

	k=2		k=3		k=4		k=5	
	Without Timing Information	With Timing Information	Without Timing Information	With Timing Information	Without Timing Information	With Timing Information	Without Timing Information	With Timing Information
BPIC2013	0.951	0.998	0.952	0.961	0.970	0.972	0.979	0.981
BPIC2020	0.985	0.998	0.986	0.998	0.986	0.998	0.989	0.998
Hospital Billing	0.785	0.998	0.973	0.995	0.945	0.957	0.945	0.997
Review Process	0.898	0.956	0.915	0.957	0.925	0.963	0.939	0.962
Road Traffic	0.768	0.976	0.823	0.976	0.963	0.979	0.969	0.975

Table 5.3: Weighted average fitness of process models generated by real life event logs

5.5.3 Process Model Discovery Performance Improvement with Timing Information

This set of experiments is to evaluate whether adding the timing information in constructing the vectors can improve process scenario discovery performance. More specifically, given an event log, we apply the proposed approach to construct the aggregated vectors that contain both activity information and timing information. For the same event log, we also apply the heuristic mining approach presented in [61] to obtain the activity vectors that contain only the activity information. For both aggregated vectors and activity vectors, we apply the k-means algorithm to partition the event logs into k clusters, where k is from 2 to 5, and discover a process model from each subset of logs using the heuristic mining algorithm presented in [80].

We use the pm4py tool [8] with the following settings: (1) process mining algorithm is heuristic mining; (2) the dependency threshold of the algorithm is 0.9; (3) the minimum number of occurrences of an activity is 1; (4) the minimum number of occurrences of an edge is 1; and (5) the thresholds for the loops of length is 2. The weighted average fitness, the weighted average precision, and the $F1$ score of the process models are depicted in Table 5.3, Table 5.4, and Table 5.5, respectively.

From the experiment results, it is clear that when we add timing information into the vector, we are able to achieve higher weighted average fitness, weighted average precision, and $F1$

	k=2		k=3		k=4		k=5	
	Without Timing Information	With Timing Information	Without Timing Information	With Timing Information	Without Timing Information	With Timing Information	Without Timing Information	With Timing Information
BPIC2013	0.816	0.899	0.721	0.857	0.675	0.804	0.651	0.759
BPIC2020	0.943	0.953	0.876	0.941	0.876	0.907	0.989	0.996
Hospital Billing	0.883	0.917	0.792	0.890	0.870	0.907	0.863	0.944
Review Process	0.530	0.754	0.431	0.571	0.475	0.551	0.492	0.646
Road Traffic	0.640	0.729	0.606	0.729	0.561	0.719	0.561	0.708

Table 5.4: Weighted average precision of process models generated by real life event logs

	k=2		k=3		k=4		k=5	
	Without Timing Information	With Timing Information	Without Timing Information	With Timing Information	Without Timing Information	With Timing Information	Without Timing Information	With Timing Information
BPIC2013	0.879	0.946	0.821	0.906	0.796	0.880	0.782	0.855
BPIC2020	0.963	0.975	0.927	0.969	0.927	0.950	0.940	0.964
Hospital Billing	0.831	0.956	0.873	0.939	0.906	0.948	0.902	0.959
Review Process	0.667	0.843	0.585	0.716	0.627	0.701	0.646	0.733
Road Traffic	0.698	0.835	0.688	0.837	0.709	0.829	0.709	0.819

Table 5.5: $F1$ score of process models generated by real life event logs

score, under different k and with different real-life event logs.

For each event log, we also calculate the average improvement percentage of different k values which is shown in Table 6.1. We observe that the improvements of the weighted average precision is better than the weighted average fitness in general. The reason is that the weighted average fitness resulted from the approach in [61] is closer to 1 than the weighted average precision, which indicates the room for fitness improvement is relatively small.

	Weighted Average Fitness	Weighted Average Precision	F1 Score
BPIC2013	2.02%	16.00%	9.53%
BPIC2020	1.24%	4.09%	2.71%
Hospital Billing	10.18%	8.06%	9.06%
Review Process	5.09%	30.30%	20.12%
Road Traffic	15.84%	20.81%	18.73%

Table 5.6: Average improvement percentage

5.5.4 Distance-based Scenario Discovery Compared with Exhaustive Search for k with k-means clustering

The third set of experiments compare the performance of the distance-based algorithm with the k-means approach. For both the proposed approach and the optimal k-means solution, we use the aggregated vectors when partitioning the event log.

To obtain the optimal k-means solution, we use the brute-force approach as follow. We apply the k-means algorithm to partition the event log to a set of sub logs with the k starting from 2 with incremental step of 1 until one of the resulted subsets becomes empty. For each resulted cluster set, we apply the heuristic mining algorithm to generate the process models, and calculate the $F1$ score, and the average $F1$ of all resulted cluster sets from k-means approach when k varies. Both the largest and the average $F1$ scores are compared with the value obtained with the proposed solution. The results are depicted in Figure 5.7.

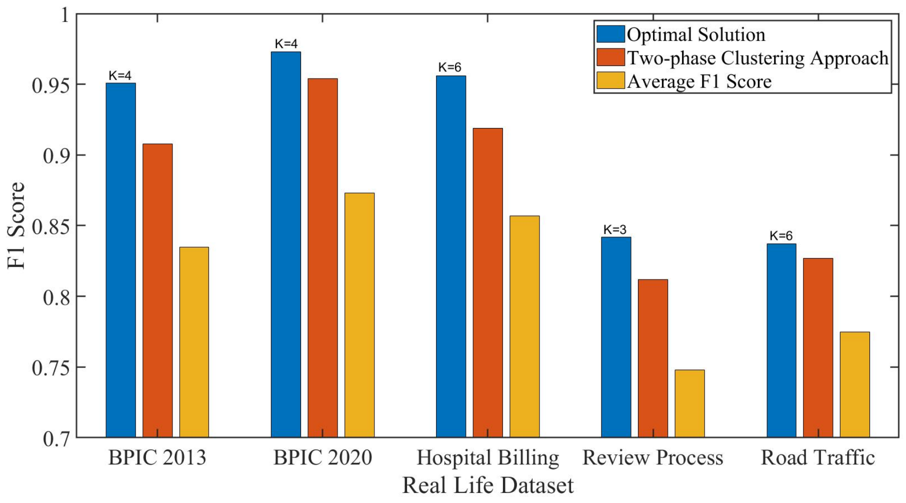


Figure 5.7: $F1$ scores of two-phase clustering approach, optimal k-means approach, and average $F1$ score of k-means approach

As shown in Figure 5.7, the proposed two-phase scenario discovery approach outperforms the k-means approach on average cases. More specifically, the $F1$ score of the two-phase approach is 7.73%, 8.21%, 7.15%, 7.46%, and 8.10 % higher than the average $F1$ score

	BPIC 2013			BPIC 2020			Review Process			Toad Traffic			Hospital Billing		
	f^W	p^W	F1 Score	f^W	p^W	F1 Score	f^W	p^W	F1 Score	f^W	p^W	F1 Score	f^W	p^W	F1 Score
Proposed	0.985	0.877	0.928	0.998	0.887	0.939	0.951	0.646	0.769	0.974	0.719	0.827	0.936	0.793	0.859
KM	0.951	0.816	0.878	0.985	0.883	0.931	0.898	0.530	0.667	0.768	0.640	0.698	0.785	0.792	0.788
DB	0.952	0.721	0.821	0.986	0.876	0.928	0.915	0.431	0.586	0.823	0.606	0.698	0.973	0.774	0.862
CP	0.965	0.675	0.794	0.986	0.872	0.926	0.925	0.475	0.628	0.963	0.561	0.709	0.945	0.619	0.748
MB	0.972	0.651	0.780	0.989	0.889	0.936	0.939	0.492	0.646	0.969	0.561	0.711	0.931	0.663	0.774

Table 5.7: Weighted average fitness, weighted average precision, and F1 score generated by process scenarios discovery approaches

for *BPIC 2013*, *BPIC2020*, *Hospital Billing*, *Review Process*, and *Road Traffic* event logs, respectively. Compared with the optimal k-means solution, the two-phase approach results in lower $f1$ score, but the difference is less than 5%.

5.5.5 Density-based Scenario Discovery Compared with Four Commonly Used Process Scenarios Discovery Approaches

This set of experiments is to compare our proposed approach with the four commonly used process scenarios discovery approaches such as the *k-means* approach (**KM**) [61], the *Model-based* approach (**MB**) [25], the *Distanced-based* approach (**DB**) [34] and the *Conserved Patterns* approach (**CP**) [16] using real-life event logs.

For each real-life event logs, we use the different approaches to generate subsets of event logs where event traces in the same subset are most likely under the same scenarios. For each subset of logs produced by the different approaches, we apply the inductive mining algorithm to discover a process model, and calculate the weighted average fitness, the weighted average precision, and the $F1$ score of the process models. The results are depicted in Table 5.7. From the experiment results, the performance of the proposed approach is 3.31%, 7.93%, and 5.61% higher than the commonly approaches with respectively for *fitness*, *precision*, and *F1 score* on average. The proposed method outperforms four commonly used approaches with different real-life event logs.

5.6 Summary

In this chapter, we present an approach that uses timing information to assist in discovering process scenarios from event logs without a prior knowledge about the number of scenarios k. The algorithm is implemented based on the open-source process mining framework *pm4py*. A real wastewater treatment process provided by a domain expert is used as a case study to investigate the effectiveness and validity of the approach. The evaluation results show that the algorithm is able to discover the process scenarios from event logs. The experiments on real-life event logs show that the process scenario models obtained with the additional timing information have higher fitness and precision scores than the models obtained without the timing information. These results lead to the conclusions that timing information can improve process scenario discovery performance, and the proposed approach can discover different process scenarios more effectively. The experiments on real-life even logs also show that the distance-based algorithm results in higher $F1$ scores on average compared to the k-means approach, and less than 5% lower $F1$ score compared with the optimal k obtained through exhaustive search.

Chapter 6

Incorporating Domain Knowledge for Implicit Knowledge Discovery

6.1 Background and Related Work

In the previous Chapter, process scenario discovery techniques have proven valuable in multiple contexts. The techniques' largest drawback is that most algorithms achieve this by solely using an event log without allowing the domain expert to influence the discovery in any way. However, the domain expert has specific domain expertise that should be exploited to create better process models in model fitness and precision criteria.

Although the field of process scenario discovery has matured in recent years, applying domain knowledge for discovering better process scenario models is still in its nascent stages. In [55], authors suggest an approach to discover a control flow model based on event logs and prior knowledge specified in terms of augmented Information Control Nets. [75] proposes a discovery algorithm using Integer Linear Programming based on the theory of regions, which can be extended with a limited set of user-specified constraints during process discovery.

In [33], the authors introduce a process discovery technique presented as a multi-relational classification problem on event logs. Their approach is supplemented by Artificially Generated Negative Events, with the possibility to include prior knowledge during discovery. Their work [55, 75, 33] addresses the issue of incorporating domain knowledge to improve the discovered process model. However, these works also face challenges in environments with different scenarios. In [22], authors propose a novel semi-supervised trace clustering technique based on expert knowledge. Their approach is based on incorporating expert knowledge starting from a complete expert solution, in which the expert provides a complete clustering solution based on the expert's expectations. However, domain experts cannot provide complete solutions in some specific domains due to the system's complexity.

In this Chapter, we present an approach that incorporates domain knowledge to assist in discovering process scenarios from event logs. Below, we provide an overview of the approach and the four steps required. First, we construct aggregated vectors for a given event log using the approach proposed in Chapter 5.3. Next, we use the density-based approach to generate an augmented ordering of the event log representing its density-based clustering structure. This augmented ordering contains information equivalent to the density-based clusterings that correspond to a broad range of parameter settings. Thirdly, we replay the original event log using the process model provided by the domain expert and obtain an expert sublog. The expert sublog shows us where the event traces fit within the process model provided by the domain expert. Based on the expert sublog, we identify the maximum similarity distance. Finally, we discover process scenarios by the maximum similarity distance and the augmented ordering.

This Chapter is organized as follows. In Chapter 6.2, we propose an algorithm to generate the augmented ordering. Next, we develop an approach to discover the process scenarios using domain knowledge and the augmented ordering in Chapter 6.3.2. Chapter 6.4 investigates the approach's effectiveness using a real wastewater treatment process and evaluates the proposed approach in terms of three criteria, i.e., the weighted average fitness, the weighted average precision, and $F1$ score using real life even logs, and discusses our findings.

6.2 Generate Augmented Ordering with a Density-based Approach

The key idea of density-based clustering is that for each event trace of a cluster, the neighborhood of a given radius(ε) has to contain at least a minimum number of event traces(*MinPts*), i.e., the cardinality of the neighborhood has to exceed a threshold. If we directly apply the existing density-based approach, i.e., DBSCAN algorithm [28], to partition the event log, we will meet a problem, which is the intrinsic cluster structure cannot be characterized by global density parameters. Different local densities may be needed to reveal clusters in a different subset of the event logs. In order to overcome this problem, we propose an algorithm that produces a special order of the event log with respect to its density-based clustering structure containing information about every clustering level of the event traces. Moreover, event traces of the event log are ordered such that spatially closest event traces become neighbors in the ordering. Additionally, the distance information is stored for each event trace representing the density that must be accepted for a cluster so that both event traces belong to the same cluster. This distance information consists of two values for each event trace: the *core-distance* and a *reachability-distance*, introduced in the following definitions.

Definition 21 (Core-distance of an aggregated vector $\vec{\mathcal{V}}$)**.** *Given an aggregated vector* $\vec{\mathcal{V}}$ *corresponding to an event trace* σ *from an event log* log, *let* ε *be a distance value, let* $N_\varepsilon(\vec{\mathcal{V}})$ *be the* $\varepsilon-$*neighborhood of* $\vec{\mathcal{V}}$*, let MinPts be a natural number and let MinPts-distance(*$\vec{\mathcal{V}}$*) be the distance from* $\vec{\mathcal{V}}$ *to its MinPts's neighbor. Then, the core-distance of an aggregated vector*

$\vec{\mathcal{V}}$ *is defined as* $CD_{\varepsilon,MinPts}(\vec{\mathcal{V}}) =$

$$\begin{cases} UNDEFINED, & if |N_\varepsilon(\vec{\mathcal{V}})| < MinPts \\ \textit{MinPts-distance}(\vec{\mathcal{V}}), & Otherwise \end{cases}$$

Consider **Trace** $1(\sigma_1)$ in Table 3.1, the aggregated vector corresponding to σ_1 is $\vec{\mathcal{V}}_1 = (1,1,1,1,0,129,130,175,174,0,0)$. Assume the *MinPts* values is 2, and the distance value is 500, i.e., *MinPts* $= 2.0$ and $\varepsilon = 500.0$. Since the *Euclidean* distances of two aggregated vectors in L are $(\vec{\mathcal{V}}_1,\vec{\mathcal{V}}_2) = 166.39, (\vec{\mathcal{V}}_1,\vec{\mathcal{V}}_3) = 237.58, (\vec{\mathcal{V}}_1,\vec{\mathcal{V}}_4) = 108.03, (\vec{\mathcal{V}}_1,\vec{\mathcal{V}}_5) = 343.18$, the core-distance of an aggregated vector $\vec{\mathcal{V}}_1$ is $CD_{500,2}(\vec{\mathcal{V}}_1) = 108.03$.

The core-distance of an aggregated vector $\vec{\mathcal{V}}$ corresponding to an event trace σ is simply the smallest distance ε between $\vec{\mathcal{V}}$ and an aggregated vector in its ε-neighborhood such that $\vec{\mathcal{V}}$ would be a core object with respect to ε if this neighbor is contained in $N_\varepsilon(\vec{\mathcal{V}})$. Otherwise, the core-distance is UNDEFINED.

Definition 22 (Reachability-distance of an aggregated vector $\vec{\mathcal{V}}_1$ and $\vec{\mathcal{V}}_2$)**.** *Given two aggregated vector* $\vec{\mathcal{V}}_1$ *and* $\vec{\mathcal{V}}_2$ *corresponding to two event traces* σ_1 *and* σ_2 *from an event log* log*, let* ε *be a distance value, let* $N_\varepsilon(\vec{\mathcal{V}}_2)$ *be the* $\varepsilon-$*neighborhood of* $\vec{\mathcal{V}}_2$*, let MinPts be a natural number Then, the reachability-distance of an aggregated vector* $\vec{\mathcal{V}}_1$ *and* $\vec{\mathcal{V}}_2$ *is defined as* $RD_{\varepsilon,MinPts}(\vec{\mathcal{V}}_1,\vec{\mathcal{V}}_2) =$

$$\begin{cases} UNDEFINED, & if |N_\varepsilon(\vec{\mathcal{V}}_2)| < MinPts \\ max(CD_{\varepsilon,MinPts}(\vec{\mathcal{V}}_2), distance(\vec{\mathcal{V}}_1,\vec{\mathcal{V}}_2)), & Otherwise \end{cases}$$

Algorithm 6 shows the procedure to create an ordering of aggregated vectors corresponding to event traces from the event log, additionally storing the core-distance and a suitable reachability-distance for each aggregated vector. The algorithm's time complexity is $O(N \times log(N))$, where N is the number of event traces in the event log L. We use Example 6 to illustrate Algorithm 6.

Algorithm 6 Generating an Augmented Ordering

```
Require: A set of aggregated vectors $\Pi = \{\vec{\mathcal{V}}_1, \cdots, \vec{\mathcal{V}}_{|\log|}\}$, a distance value $\varepsilon$ , and a natural
    number $MinPts$.
Ensure: An ordering of aggregated vectors and corresponding reachability-distance for each
    aggregated vector
 1: for each aggregated vectors $\vec{\mathcal{V}} \in \Pi$ do
 2:   Set $\vec{\mathcal{V}}$'s reachability distance is UNDEFINED
 3: end for
 4: for each unprocessed aggregated vectors $\vec{\mathcal{V}} \in \Pi$ do
 5:   $N$ is the $\varepsilon-$neighborhood of $\vec{\mathcal{V}}$
 6:   mark $\vec{\mathcal{V}}$ as processed
 7:   output $\vec{\mathcal{V}}$ to the ordered list
 8:   if $CD_{\varepsilon,MinPts}(\vec{\mathcal{V}}) \neq UNDEFINED$ then
 9:     Seeds = empty priority queue
10:     for each next $q$ in Seeds do
11:       M is the $\varepsilon-$neighborhood of $q$
12:       mark $q$ as processed
13:       output $q$ to the ordered list
14:       if $CD_{\varepsilon,MinPts}(q) \neq UNDEFINED$ then
15:         coredist = $CD_{\varepsilon,MinPts}(q)$
16:         for each $o$ in M do
17:           if $o$ is not processed then
18:             reachDist = max(coredist, dist(q,o))
19:             if $RD_{\varepsilon,MinPts}(o,q) \neq UNDEFINED$ then
20:               $Seeds.insert(o, RD_{\varepsilon,MinPts}(o,q))$
21:             else
22:               $Seeds.move(o, RD_{\varepsilon,MinPts}(o,q))$
23:             end if
24:           end if
25:         end for
26:       end if
27:     end for
28:   end if
29: end for
```

Example 6. *Consider the set of aggregated vectors* $\Pi = \{\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4, \vec{\mathcal{V}}_5\}$ *given in the Table 3.1, and assume the two parameters are MinPts* $= 2.0$ *, and* $\varepsilon = 500.0$*, the corresponding core-distance of each aggregated vector in the set*Π *are* $CD_{500,2}(\vec{\mathcal{V}}_1) = 108.03$, $CD_{500,2}(\vec{\mathcal{V}}_2) = 91.28$, $CD_{500,2}(\vec{\mathcal{V}}_3) = 237.58$, $CD_{500,2}(\vec{\mathcal{V}}_4) = 91.28$, $CD_{500,2}(\vec{\mathcal{V}}_5) = 306.72$.

According to Line 2-4 of algorithm, we select an aggregated vector $\vec{\mathcal{V}}_1$*, and check the label of aggregated vector* $\vec{\mathcal{V}}_1$*. We apply Line 8-14 of the algorithm for every aggregated vector in the set* Π *to find the neighborhood of aggregated vector* $\vec{\mathcal{V}}_1$*. Since the distance between aggregated vectors* $\vec{\mathcal{V}}_1$ *and* $\vec{\mathcal{V}}_2$*, i.e.* $d(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2)$*, is* 108.03 *and is less than* ε*, we add* $\vec{\mathcal{V}}_1$ *into priority queue Seeds. After applying the step to the other three aggregated vectors* $\vec{\mathcal{V}}_2, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_4, \vec{\mathcal{V}}_5$*, the Seeds becomes* $\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2, \vec{\mathcal{V}}_4, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_5$ *in order of distance from smallest to largest. Then we remove the first element* $\vec{\mathcal{V}}_1$ *from the priority queue Seeds and set the* $\vec{\mathcal{V}}_1$ *as the point* p *on Line 10-15. Based on Line 17- 19, we calculate the reachability-distance of the aggregated vector* $\vec{\mathcal{V}}_1$ *and other aggregated vectors in priority queue Seeds, i.e.*$RD_{500,2}(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2) = 91.28, RD_{500,2}(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_3) = 237.58, RD_{500,2}(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_4) = 91.28, RD_{500,2}(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_5) = 306.72$*, and then find the smallest reachability-distance, i.e.* $RD_{500,2}(\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2) = 91.28$.

We apply Line 20-22 of the algorithm to remove the aggregated vector, which has the smallest reachability-distance in the queue Seeds, and the Seeds becomes $\vec{\mathcal{V}}_4, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_5$*. We set the* $\vec{\mathcal{V}}_2$ *as the point* p *and repeat the steps until all aggregated vectors are labeled. Finally, the ordering of aggregated vectors is* $\vec{\mathcal{V}}_1, \vec{\mathcal{V}}_2, \vec{\mathcal{V}}_4, \vec{\mathcal{V}}_3, \vec{\mathcal{V}}_5$ *and the corresponding reachability-distances are* $108.03, 91.28, 91, 28, 237.58, 306.72$

6.3 Discover Process Scenarios Based on Domain Knowledge and the Augmented Ordering

This chapter presents an approach to discovering the process scenarios from event logs based on domain knowledge. This approach has two steps. First, we identify the maximum similarity distance in the event logs based on domain knowledge. Second, we use the approach on the maximum similarity distance and the augmented ordering produced by algorithm 6 to generate subsets of event logs where event traces in the same subset are most likely under the same scenarios.

6.3.1 Identify the Maximum Similarity Distance in Event Logs Based on Domain Knowledge

Before we introduce the approach to identify the maximum similarity distance in the event log based on the domain knowledge, we first discuss how expert knowledge can be represented and how it can be incorporated into a trace clustering approach. Typically, domain knowledge can be represented in three distinct approaches: expert-driven initialization, constraints clustering, and process model initialization [22].

The expert-driven initialization approach is related to semi-supervised learning [10]. In this approach, the domain expert is expected to manually assign a small subset of traces to a cluster, after which a clustering algorithm extends the clusters to the entire event log. The approach is useful for centroid-based algorithms such as k-means algorithms, which often rely on random initialization of starting data points in order to commence the clustering. By setting these starting data points based on the domain knowledge of an expert instead of randomly, the performance of the trace clustering results should increase.

The constraints clustering approach refers to specify domain knowledge effectively in terms of constraints. Rather than provide a starting subset of clustered traces, the expert provides a set of constraints to which the clustering solution is expected to conform. Typical examples of expert constraints are *must-link* constraints, which indicate that two elements must be included in the same cluster, and *cannot-link* constraints, indicating that two elements should not be clustered together [78].

The process model initialization approach uses process model as a way to input the domain knowledge based on the expert's experience. For the process model initialization approach, the expert provides a process model representing the primary process scenario across the entire event log. In this case, a clustering obtained with the use of the process model could be a useful starting point for a clustering exercise. Then, the solution of the expert and the regular trace clustering solution can be combined to create a consensus clustering. We use the process model to input the domain knowledge in this work. The expert-driven initialization approach and the constraints clustering approach, it will be addressed in our future work.

Given an event log and the process model provided by the domain expert, the maximum similarity distance refers to the maximum *Euclidean* distances between aggregated vectors in the primary process scenario, which is in the form of the process model provided by the domain expert. Our approach to identify the maximum similarity distance consists of two steps: (1) obtaining the primary process scenario sublog by replaying the original event log on the primary process scenario model; (2) identifying the maximum similarity distance based on the primary process scenario sublog.

For event traces in the original event log, we identify whether an event trace is replayable defined in Chapter 3.2 on the primary process scenario model . We refer the set of replayable event traces as the primary process scenario sublog. Let $d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j)$ denote *Euclidean* distances between aggregated vectors $\vec{\mathcal{V}}_i$, $\vec{\mathcal{V}}_j$. Algorithm 7 shows the procedure to identify the maximum similarity distance from aggregated vectors based on the domain knowledge. The time complexity of the algorithm is $O(N \times logN)$, where N is the number of event traces in the event log L. We use Example 7 to illustrate Algorithm 7.

Example 7. *Consider the event log given in Table 3.1 and a process model shown in Figure 6.1 provided by a domain expert. According to Line 2-7 of algorithm, we select four event traces from the event log based on the Definition 10, convert them into the corresponding aggregated vectors, and put them into the set A. Hence, the set A is updated to* $\{(1, 1, 1, 1, 0, 129, 130, 175, 174, 0, 0), (1, 1, 1, 1, 0, 202, 122, 37, 117, 0, 0), (1, 1, 1, 1, 0, 54, 279, 245, 20, 0, 0), (1, 1, 1, 1, 0, 178, 92, 87, 183, 0, 0)\}$. *We apply Line 8-13 of algorithm for every*

Algorithm 7 IDENTIFY THE MAXIMUM SIMILARITY DISTANCE FROM AGGREGATED VECTORS

```
Require: a workflow net $N = (P, T, F)$ provided by the domain expert, a event log $L$
Ensure: the Maximum Similarity Distance $msd$
 1: Define a set $A = \{\}$ ;
 2: for each event trace $\sigma \in L$ do
 3:     if $\sigma$ is replayable on the $N$ then
 4:         Convert the $\sigma$ to aggregated vector $\vec{\mathcal{V}}$
 5:         $A = A \cup \{\vec{\mathcal{V}}\}$
 6:     end if
 7: end for
 8: Set $msd = 0$
 9: for each two aggregated vector $\vec{\mathcal{V}}_i$, and $\vec{\mathcal{V}}_j \in A$ do
10:     if $d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j) \geq M$ then
11:         $msd = d(\vec{\mathcal{V}}_i, \vec{\mathcal{V}}_j)$
12:     end if
13: end for
14: return $msd$
```

aggregated vector in the the set A. After applying the procedure, the maximum similarity distance produced by the event trace σ_2 and event trace σ_3 is 315*, i.e.* $msd = 315$.

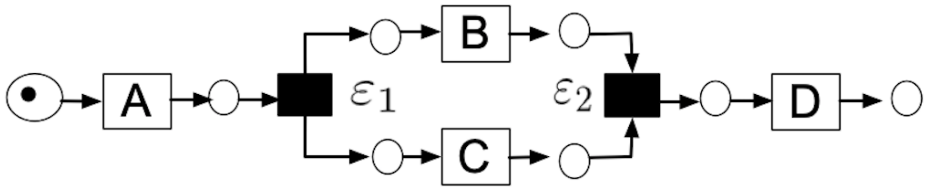


Figure 6.1: Process model representing the domain knowledge

6.3.2 Discover Process Scenarios Based on the Maximum Similarity Distance and the Augmented Ordering

The algorithm mentioned in Chapter 6.2 generates the augmented ordering consisting of the ordering of the event traces and the corresponding reachability-distance. In the following, we propose an approach using the maximum similarity distance and the augmented ordering to generate subsets of event logs where event traces in the same subset are most likely under the same scenarios. To simplify the notation, we specify the augmented ordering formally:

Definition 23 (Augmented Ordering)**.** *Given an event log L, a distance value ε, and a natural number MinPts, the augmented ordering Λ is the result produced by the Algorithm 6 $[\lambda_1, \dots \lambda_i]$ where λ_i is characterized by a 3-tuple$(m, n, RD_{\varepsilon,MinPts})$, the m is index of the augmented order, the n is the index of the event trace, and the $RD_{\varepsilon,MinPts})$ is reachability-distance of the the corresponding event traceσ_n* □

Algorithm 8 Discover process scenarios based on the maximum similarity distance and the augmented ordering

```
Require: a event log $L$, the Maximum Similarity Distance $msd$, and the the augmented
    ordering $\Lambda = [\lambda_1, \dots \lambda_i]$, where $\lambda_i = (m, n, RD_{\varepsilon,MinPts})$
Ensure: subsets of event logs
 1: Define two sets $Subsets = \{\}$ and $\Delta = \{\}$;
 2: for each $\lambda_i \in \Lambda$ do
 3:   if $\lambda_i.RD_{\varepsilon,MinPts} \leq msd$ then
 4:     $\Delta = \Delta \cup \{\lambda_i.n\}$
 5:     CONTINUE
 6:   end if
 7:   if $\lambda_i.RD_{\varepsilon,MinPts} > msd$, and $\Delta \neq \{\}$ then
 8:     $Subsets = Subsets \cup \Delta$
 9:     $\Delta = \{\}$
10:   end if
11: end for
12: return $Subsets$
```

Based on the Definition 23, we know the augmented ordering is a special order of all the event traces in the event log with respect to its density-based clustering structure containing the information about every clustering level of the event traces. The Algorithm 6 does not assign cluster memberships. Hence, we apply the maximum similarity distance to assign cluster memberships. Algorithm 8 gives the detailed steps of generating subsets of event logs where event traces in the same subset are most likely under the same scenarios. The time complexity of Algorithm 8 is $O(N)$, where N is the number of events in a trace. Finally, the existing process discovery algorithms can be applied on the subset of logs to obtain different process scenarios in terms of process models.

6.4 Experimental Study

In this Chapter, we first apply the proposed technique to the artificial logs to investigate the effectiveness and validity. Then we experimentally evaluate the proposed approach on five real-life event logs shown in Chapter 3.4. The objectives of our evaluations consist of two components. First, investigate the effectiveness of process scenario discoveries in real wastewater treatment processes. To do so, we apply the proposed approach on the artificial logs generated from the **Plg** based on the wastewater treatment process and a process model provided by a domain expert to examine the effectiveness of the approach. Second, we evaluate if incorporating domain knowledge can improve the quality of process scenario models discovered from event logs. In order to achieve the objective, we choose an existing heuristic mining algorithm [81], and compare the scenario models obtained by the proposed approach with the commonly used k-means clustering approaches where k is exhaustively searched.

6.4.1 Effectiveness Validation in Wastewater Treatment Domain

This set of experiments examines the proposed approach's effectiveness for process scenario discovery in the wastewater treatment domain. More specifically, based on the wastewater treatment process, we first use the **Plg** to generate five artificial event logs by injecting an incremental amount of noise ranging from 1% to 20% to produce an event log that contains one thousand event traces. Then, for each artificial event log, based on the process model provided by the domain expert, we apply the proposed approach in Chapter 6.3.2 to partition the event log into clusters using the process model provided by the domain expert, and discover a process model from each subset of the event log using the heuristic mining algorithm presented in [81].

Figure 5.3 shows the entire wastewater treatment process in the form of the Business Process Modeling Notation (**BPMN**) [85]. All wastewater treatment processes begin with influent wastewater from sewage lines arriving at a treatment facility. The wastewater is sent through coarse and fine screens to remove large and small objects, debris, etc., from the water. After these first mandatory steps are completed, the wastewater will go through different sub-processes depending on if the water can be reused or if it will be disposed of at a landfill.

The process model representing the primary process scenario provided by the domain expert is shown in Figure 6.1. This scenario describes the liquid flow through the wastewater treatment process. The process starts with coarse and fine screen activities and then goes to the "Primary Clarifiers" activity to allow solids to separate from the liquids. The liquid then moves to the "Bio-reactor Tanks," where the liquid is treated by microfiltration membrane bioreactors (MBRs) that combine a suspended growth biological reactor with microfiltration for secondary water treatment. Then, depending on various chemical levels, the water will travel to the "Final Clarifier" activity, where an activated sludge process is used to settle out microorganisms. Then the "Chlorine Contact Basin" is also used to disinfect the water and remove microorganisms, bacteria, viruses, etc., from the water. Water is then moved to "Ponds," where organic content and pathogens are reduced from the wastewater. Then the "De-chlorination Pond" is used to reduce the chlorine levels in the wastewater. Before the process ends, the water is moved to either the "Effluent Pumping Station" for transporting the water or the Cascade Aerator to oxidize iron and reduce dissolved gases.

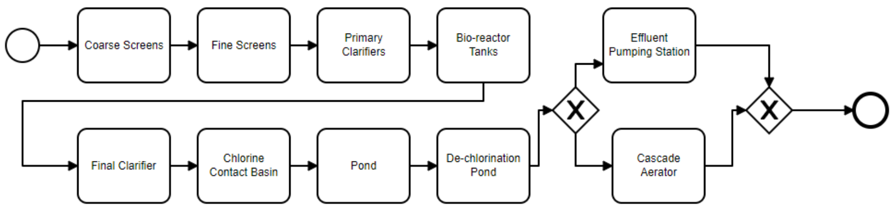


Figure 6.2: Process model related to the primary process scenario provided by the domain expert

Figure 6.3, Figure 6.4, and Figure 6.5 show the results produced by the proposed approach. The first scenario within the wastewater treatment process shown in Figure 6.3 is used to remove solid debris in the water. This process scenario begins at the "Coarse Screens" activity. These screens remove the larger solids, such as rags, sticks, rocks, etc. After the large debris is removed from the wastewater, it moves to the "Fine Screens" activity, where smaller particles are moved from the water. The final processing step is the "Compacted Screenings," where the previously used screens are washed. They are then compacted to reduce the amount of water and increase the concentration of solids. The first scenario ends with the "Disposal to Landfill" activity where waste is disposed of.

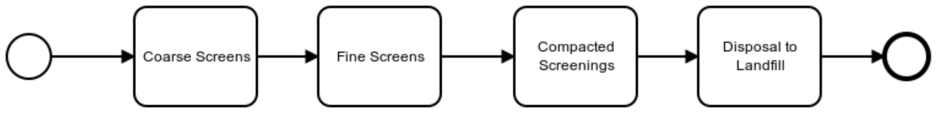


Figure 6.3: Removing solid debris process in wastewater treatment process

The second scenario within the wastewater treatment process shown in Figure 6.4 is used to treat sludge by converting to a solid form. The same process of water going through the coarse and fine screens occurs as in the first scenario. Then, the water goes through the "Primary Clarifiers" activity, where solids are separated from the wastewater by gravity in a clarification tank. After this activity, "Blending" routes some of the water flow away from the treatment process so that it can be blended with fully treated water before it is disposed of. The "Anaerobic Digester" step continues the treatment process by degrading organic matter by removing oxygen. The process can either end at this point or continue with the "Storage Tank" activity, where water can be stored before it is disposed of at a landfill.

The third scenario within the wastewater treatment process shown in Figure 6.5 describes the liquid flow through the process. The third scenario also starts with coarse/fine screen activities. It then goes to the "Primary Clarifiers" and the "Bio-reactor Tanks" activities, similar to the process model representing the primary process scenario provided by the domain expert, to allow solids to separate from the liquids. The water will travel to one of

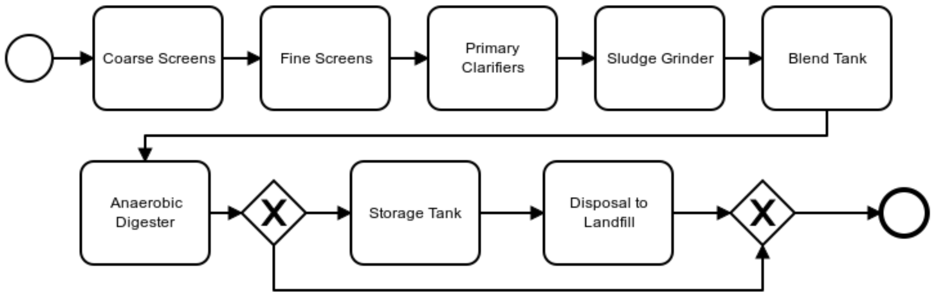


Figure 6.4: Treating sludge process in wastewater treatment process

three activities. The first, “Gravity Thickening,” is used to condense biosolids to separate the solids from the solid-free supernatant. The second activity, “Dissolved-Air Flotation Thickening”, thickens sludge to bring solids to the surface, separating the solids and liquids. The final option is “Wastewater Centrifuge De-watering Thickening,” where a high-speed rotation process separates water from the sewage. After one of these processes is used to treat the wastewater, it is then sent to the “Blend Tank,” and the “Anaerobic Digester” for further treatment before the process ends. The last scenario is the same as the primary process scenario provided by the domain expert shown in Figure 6.1. The evaluation results show that the proposed approach is able to discover the process scenarios from event logs.

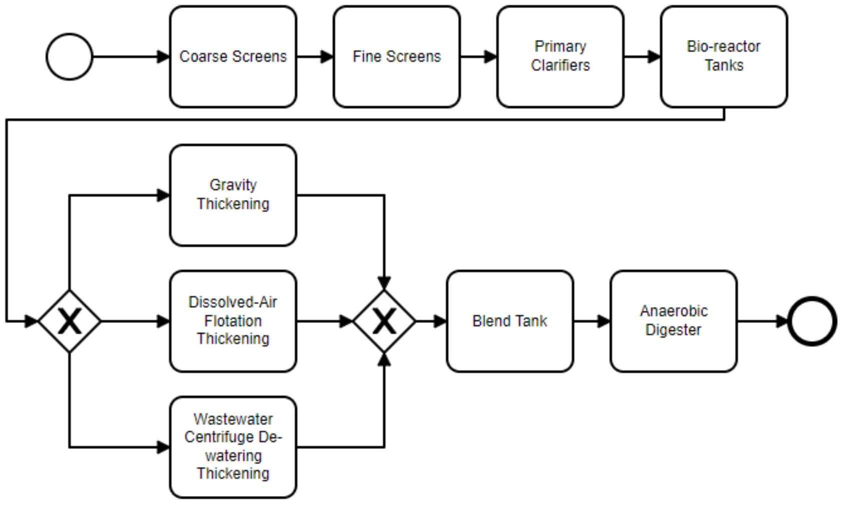


Figure 6.5: Thickening process in wastewater treatment process

6.4.2 Performance Evaluation

This set of experiments evaluates whether incorporating domain knowledge can improve the process scenario discovery performance. More specifically, given an event log, we apply the proposed approach to generate subsets of event logs where event traces in the same subset are most likely under the same scenarios. For the same event log, we also apply the trace clustering approach presented in [61] to partition the event logs into k clusters, where k is from 2 to 5. For each subset of logs produced by the different approaches, we discover a process model using the heuristic mining algorithm presented in [80].

We use the pm4py tool [8] with the following settings: (1) the process mining algorithm is heuristic mining; (2) the dependency threshold of the algorithm is 0.9; (3) the minimum number of occurrences of an activity is 1; (4) the minimum number of occurrences of an edge is 1; and (5) the thresholds for the loops of length is 2. The weighted average fitness, the weighted average precision, and the $F1$ score of the process models are depicted in Table 5.7.

From the experiment results, it is clear that when we incorporate the domain knowledge in the trace clustering, we can achieve higher weighted average fitness, weighted average precision, and $F1$ score under different real-life event logs.

For each event log, we also calculate the average improvement percentage of different k values, shown in Table 6.1. We observe that the improvements in the weighted average precision are better than the weighted average fitness in general. The reason is that the weighted average fitness resulting from the approach in [61] is closer to 1 than the weighted average precision, which indicates the room for fitness improvement is relatively small.

	Weighted Average Fitness	Weighted Average Precision	F1 Score
BPIC2013	2.02%	10.30%	12.53%
BPIC2020	1.24%	4.09%	3.71%
Hospital Billing	8.18%	8.96%	8.06%
Review Process	5.09%	7.30%	12.12%
Road Traffic	3.84%	9.81%	11.73%

Table 6.1: Average improvement percentage under real life event logs

6.5 Summary

This chapter presents a density-based trace clustering technique based on expert knowledge to assist process scenario discovery results. In order to incorporate domain knowledge in process scenario discoveries, we first generate an augmented ordering of the event log representing its density-based clustering structure. Then, we identify the maximum similarity distance using the process model provided by the domain expert. Finally, we partition the event log into a set of sublogs by the maximum similarity distance and the augmented ordering. The algorithm is implemented based on the open-source process mining framework *pm4py*. A real wastewater treatment process is used as a case study to investigate the effectiveness and validity of the approach. The evaluation results show that the algorithm can discover the process scenarios from event logs in the wastewater treatment domain. The experiments on real-life event logs show that the proposed approach results in a higher fitness, precision, and F1 scores on average compared to the k-means approach.

Chapter 7

Conclusion

Although automation has become widespread in many industries, some workplaces, such as utility industries, still rely heavily on individuals to perform critical tasks based on their extensive past experiences. This accumulated intellectual capital not only improves productivity and efficacy under normal conditions but, more importantly, also contains knowledge about identifying anomalies and addressing unexpected events and situations. This knowledge is essential and critical to ensuring system safety and reliability, especially for industries with aging infrastructures where anomalies are becoming more common. Unfortunately, this knowledge is not explicitly recorded in defined guidelines, protocols, or standard workflows but implicitly resides in the minds of skilled workers and routine event logs. As skilled workers retire or leave the business, we may lose this accrued indispensable knowledge that is critical to the industries' productivity, reliability, and safety. Hence, this thesis studies the issues of implicit knowledge and develops a framework for discovering implicit knowledge from event logs. The implementation of the framework contains three phases. We first present the design and implementation of an approach that uses hidden Markov models to filter out outliers from event logs in the cleaning phase. For the discovery phase, we propose approaches to discover the implicit knowledge related to timing constraints and process scenarios from event logs. This phase takes the clean event logs as input. The implicit knowledge related to timing constraints or process scenarios is output represented by process models. In the incorporating phase, we present an approach incorporating domain knowledge to discover implicit knowledge related to process scenarios from event logs to improve performance in terms of fitness, precision, and F1 value. The case studies and experiment results indicate that the proposed implicit knowledge discovery framework can mine implicit knowledge related to the timing constraints and process scenarios from event logs and outperform commonly used approaches with different real-life event logs.

Beyond the scope of this thesis, there are several open challenges.

1. In the future, the proposed framework for discovering implicit knowledge needs to be augmented to handle inaccurate information, misleading prior experience or lack of experience, domain language barrier, and large amounts of data and ensure effectiveness in terms of correctness, preciseness, and timeliness.

2. Issue reporting and resolution is a software engineering process supported by tools such as the Issue Tracking System (ITS), Peer Code Review (PCR) system, and Version Control System (VCS) [37]. Several open-source software projects follow a process in which a defect or feature enhancement request is reported to an issue tracker, followed by source-code change or patch review and patch commit using a version control system. We can apply the proposed framework to three software repositories (ITS, PCR, and VCS) from a control flow perspective for effective implicit knowledge discovery. For example, we can discover a runtime process model for a bug resolution process spanning three repositories and conduct process performance and efficiency analysis.

3. Whereas the main focus of implicit knowledge discovery is on the control-flow perspective and the time perspective, event logs may contain a wealth of information relating to other perspectives, such as the organizational perspective, the case perspective, and the other

perspective. Organizational mining can be used to get insight into typical work patterns, organizational structures, and social networks. Case data can be used to understand decision-making better and analyze differences among cases. Moreover, the different perspectives can be merged into a single model providing an integrated view of the process. Such an integrated model can be used for more effective implicit knowledge analysis. In the future, we will enhance the proposed framework from different perspectives.

Bibliography

[1] Bpi challenge 2013. https://data.4tu.nl/repository/uuid:3537c19d-6c64-4b1d-815d-915ab0e479da.

[2] Bpi challenge 2020. https://data.4tu.nl/repository/uuid:3f422315-ed9d-4882-891f-e180b5b4feb5.

[3] Hospital billing - event log. https://data.4tu.nl/repository/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741.

[4] Road traffic fine management process. https://data.4tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

[5] Synthetic event logs - review example. https://data.4tu.nl/repository/uuid:da6aafef-5a86-4769-acf3-04e8ae5ab4fe.

[6] *Road Traffic Fine Management Process Data*, 2015. https://data.4tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

[7] *process-discovery*, 2016. http://pm4py.pads.rwth-aachen.de/documentation/process-discovery/alpha-miner/.

[8] *State-of-the-art-process mining in Python*, 2020. https://pm4py.fit.fraunhofer.de/.

[9] E. Abdi. Supernumerary robotic arm for three-handed surgical application: behavioral study and design of human-machine interface. Technical report, EPFL, 2017.

[10] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *In Proceedings of 19th International Conference on Machine Learning (ICML-2002*. Citeseer, 2002.

[11] L. E. Baum. Growth functions for trasformations on manifolds. *Pac. J. Math.*, 27(2):211–227, 1968.

[12] L. E. Baum, J. A. Eagon, et al. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3):360–363, 1967.

[13] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process mining based on regions of languages. In *International Conference on Business Process Management*, pages 375–383. Springer, 2007.

[14] A. Bogarín Vega, R. Cerezo Menéndez, and C. Romero. Discovering learning processes using inductive miner: A case study with learning management systems (lmss). *Psicothema*, 2018.

[15] R. J. C. Bose and W. M. Van der Aalst. Context aware trace clustering: Towards improving process mining results. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 401–412. SIAM, 2009.

[16] R. J. C. Bose and W. M. van der Aalst. Trace clustering based on conserved patterns: Towards achieving better process models. In *International Conference on Business Process Management*, pages 170–181. Springer, 2009.

[17] M. W. Browne. Cross-validation methods. *Journal of mathematical psychology*, 44(1):108–132, 2000.

[18] A. Burattin. Plg2: Multiperspective process randomization with online and offline simulations. In *BPM (Demos)*, pages 1–6, 2016.

[19] H.-J. Cheng and A. Kumar. Process mining on noisy logs—can log sanitization help to improve performance? *Decision Support Systems*, 79:138–149, 2015.

[20] R. Conforti, M. La Rosa, and A. H. ter Hofstede. Filtering out infrequent behavior from business process event logs. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):300–314, 2016.

[21] I. Craig, C. Aldrich, R. Braatz, F. Cuzzola, E. Domlan, S. Engell, J. Hahn, V. Havlena, A. Horch, B. Huang, et al. Control in the process industries. *The impact of control technology. IEEE control systems society*, 2011.

[22] P. De Koninck, K. Nelissen, B. Baesens, S. vanden Broucke, M. Snoeck, and J. De Weerdt. An approach for incorporating expert knowledge in trace clustering. In *International Conference on Advanced Information Systems Engineering*, pages 561–576. Springer, 2017.

[23] A. A. De Medeiros, B. F. van Dongen, W. M. Van der Aalst, and A. Weijters. Process mining: Extending the α-algorithm to mine short loops. 2004.

[24] A. K. A. de Medeiros, A. J. Weijters, and W. M. van der Aalst. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.

[25] J. De Weerdt, S. Vanden Broucke, J. Vanthienen, and B. Baesens. Active trace clustering for improved process discovery. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2708–2720, 2013.

[26] J. Desel and W. Reisig. *Place/transition Petri Nets*, pages 122–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[27] J. Eder, E. Panagos, H. Pozewaunig, and M. Rabinovich. Time management in workflow systems. In *BIS'99*, pages 265–280. Springer, 1999.

[28] 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.

[29] M. Fani Sani, S. van Zelst, and W. Aalst. Applying sequence mining for outlier detection in process mining: Confederated international conferences: Coopis, ctc, and odbase 2018, valletta, malta, october 22-26, 2018, proceedings, part ii. pages 98–116, 10 2018.

[30] M. Fani Sani, S. van Zelst, and W. Aalst. *Repairing Outlier Behaviour in Event Logs*, pages 115–131. 06 2018.

[31] F. Folino, G. Greco, A. Guzzo, and L. Pontieri. Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction. *Data & Knowledge Engineering*, 70(12):1005–1029, 2011.

[32] S. Goedertier, J. De Weerdt, D. Martens, J. Vanthienen, and B. Baesens. Process discovery in event logs: An application in the telecom industry. *Applied Soft Computing*, 11(2):1697–1710, 2011.

[33] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10:1305–1340, 2009.

[34] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Discovering expressive process models by clustering log traces. *IEEE Transactions on knowledge and data engineering*, 18(8):1010–1027, 2006.

[35] C. W. Günther. Process mining in flexible environments. 2009.

[36] C. W. Günther and W. M. Van Der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer, 2007.

[37] M. Gupta, A. Sureka, and S. Padmanabhuni. Process mining multiple repositories for software defect resolution from control and organizational perspective. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 122–131, 2014.

[38] J. Herbst and D. Karagiannis. An inductive approach to the acquisition and adaptation of workflow models. In *Proceedings of the IJCAI*, volume 99, pages 52–57. Citeseer, 1999.

[39] L. Holloway, C. Bear, and K. Wilkinson. Re-capturing bovine life: Robot–cow relationships, freedom and control in dairy farming. *Journal of Rural Studies*, 33:131–140, 2014.

[40] A. K. Jain and R. C. Dubes. *Algorithms for clustering data.* Prentice-Hall, Inc., 1988.

[41] K. Jensen and L. M. Kristensen. *Coloured Petri nets: modelling and validation of concurrent systems.* Springer Science & Business Media, 2009.

[42] F. Jentsch. *Human-robot interactions in future military operations.* CRC Press, 2016.

[43] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[44] M. Kuhn. *Knowledge Management in a Steel Company: A case study of the Gerdau Group.* PhD thesis, Massachusetts Institute of Technology, 2009.

[45] S. Leemans, D. Fahland, and W. Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. volume 171, pages 66–78, 05 2014.

[46] S. J. Leemans, D. Fahland, and W. M. van der Aalst. Discovering block-structured process models from event logs-a constructive approach. In *International Conference on Applications and Theory of Petri Nets and Concurrency*, pages 311–329, Berlin, Heidelberg, 2013. Springer.

[47] S. Ling and H. Schmidt. Time petri nets for workflow modelling and analysis. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 4, pages 3039–3044. IEEE, 2000.

[48] S. Louhichi, M. Gzara, and H. B. Abdallah. A density based algorithm for discovering clusters with varied density. In *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, pages 1–6. IEEE, 2014.

[49] F. Mannhardt, M. De Leoni, H. A. Reijers, and W. M. Van Der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, 2016.

[50] O. Marjanovic and M. E. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Systems*, 1(2):157–192, 1999.

[51] O. B. P. Model. Notation (bpmn). object management group, dtc. 2010.

[52] J. Qiu, F. Lü, H. Zhang, L. Shao, and P. He. Data mining strategies of molecular information for inspecting wastewater treatment by using uhrms. *Trends in Environmental Analytical Chemistry*, 31:e00134, 2021.

[53] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[54] L. A. Reid. Personal knowledge: Towards a post-critical philosophy, 1959.

[55] A. J. Rembert, A. Omokpo, P. Mazzoleni, and R. T. Goodwin. Process discovery using prior knowledge. In *International conference on service-oriented computing*, pages 328–342. Springer, 2013.

[56] A. Rozinat and W. M. Van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

[57] A. Rozinat, M. Veloso, and W. M. Van Der Aalst. Using hidden markov models to evaluate the quality of discovered process models. *Extended Version. BPM Center Report BPM-08-10, BPMcenter. org*, 161:178–182, 2008.

[58] M. F. Sani, S. J. van Zelst, and W. M. van der Aalst. Improving process discovery results by filtering outliers using conditional behavioural probabilities. In *International Conference on Business Process Management*, pages 216–229. Springer, 2017.

[59] M. F. Sani, S. J. van Zelst, and W. M. van der Aalst. Applying sequence mining for outlier detection in process mining. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 98–116. Springer, 2018.

[60] M. Solé and J. Carmona. Process mining from a basis of state regions. In *International Conference on Applications and Theory of Petri Nets*, pages 226–245. Springer, 2010.

[61] M. Song, C. W. Günther, and W. M. Van der Aalst. Trace clustering in process mining. In *International conference on business process management*, pages 109–120. Springer, 2008.

[62] M. Song and W. M. Van der Aalst. Towards comprehensive support for organizational mining. *Decision support systems*, 46(1):300–317, 2008.

[63] S. Suriadi, R. Andrews, A. H. ter Hofstede, and M. T. Wynn. Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. *Information Systems*, 64:132–150, 2017.

[64] S. Suriadi, R. S. Mans, M. T. Wynn, A. Partington, and J. Karnon. Measuring patient flow variations: A cross-organisational process mining approach. In *Asia-Pacific Conference on Business Process Management*, pages 43–58. Springer, 2014.

[65] S. Suriadi, M. T. Wynn, C. Ouyang, A. H. ter Hofstede, and N. J. van Dijk. Understanding process behaviours in a large insurance company in australia: A case study. In *International Conference on Advanced Information Systems Engineering*, pages 449–464. Springer, 2013.

[66] M. Toossi. Labor force projections to 2014: Retiring boomers. *Monthly Lab. Rev.*, 128:25, 2005.

[67] M. Toossi. Labor force projections to 2024: The labor force is growing, but slowly. *Monthly Lab. Rev.*, 138:1, 2015.

[68] W. Van Der Aalst. *Process mining: discovery, conformance and enhancement of business processes*, volume 2. Springer, 2011.

[69] W. Van Der Aalst. Data science in action. In *Process mining*, pages 3–23. Springer, 2016.

[70] 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.

[71] 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.

[72] W. M. Van Der Aalst. Three good reasons for using a petri-net-based workflow management system. In *Information and Process Integration in Enterprises*, pages 161–182. Springer, 1998.

[73] W. M. Van der Aalst, A. Weijters, and L. Maruster. Workflow mining: Which processes can be rediscovered. Technical report, BETA Working Paper Series, WP 74, Eindhoven University of Technology, Eindhoven, 2002.

[74] W. M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, Heidelberg, 2 edition, 2016.

[75] J. M. E. Van der Werf, B. F. van Dongen, C. A. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. In *International conference on applications and theory of petri nets*, pages 368–387. Springer, 2008.

[76] S. K. vanden Broucke and J. De Weerdt. Fodina: A robust and flexible heuristic process discovery technique. *decision support systems*, 100:109–118, 2017.

[77] G. M. Veiga and D. R. Ferreira. Understanding spaghetti models with sequence clustering for prom. In *International Conference on Business Process Management*, pages 92–103. Springer, 2009.

[78] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, et al. Constrained k-means clustering with background knowledge. In *Icml*, volume 1, pages 577–584, 2001.

[79] V. S. Ware and H. Bharathi. Study of density based algorithms. *International Journal of Computer Applications*, 69(26), 2013.

[80] 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.

[81] A. J. Weijters and W. M. Van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

[82] L. Wen, J. Wang, and J. Sun. Mining invisible tasks from event logs. In *Advances in Data and Web Management*, pages 358–365. Springer, 2007.

[83] L. Wen, J. Wang, W. M. van der Aalst, B. Huang, and J. Sun. Mining process models with prime invisible tasks. *Data & Knowledge Engineering*, 69(10):999–1021, 2010.

[84] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer Berlin, 2020.

[85] S. A. White. Introduction to bpmn. *Ibm Cooperation*, 2(0):0, 2004.

[86] J. Whittle, R. Kwan, and J. Saboo. From scenarios to code: An air traffic control case study. *Software & Systems Modeling*, 4(1):71–93, 2005.

[87] M. Wil Van Der Aalst and C. Stahl. *Modeling business processes: a petri net-oriented approach.* MIT press, 2011.