TECHNISCHE
UNIVERSITÄT
DARMSTADT

Intelligent Computer-assisted Process Mining

vom Fachbereich Informatik
der Technischen Universität Darmstadt

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

Dissertation

von **Alexander Niklas Seeliger, M.Sc.**

Erstreferent: Prof. Dr. Max Mühlhäuser (Technische Universität Darmstadt)

Korreferent: Prof. Dr. Michael Rosemann (Queensland University of Technology)

Tag der Einreichung: 06. Mai 2020

Tag der Disputation: 26. Juni 2020

Darmstadt 2020

Alexander Niklas Seeliger: *Intelligent Computer-assisted Process Mining*
Darmstadt, Technische Universität Darmstadt

Jahr der Veröffentlichung der Dissertation auf TUprints: 2020
URN: urn:nbn:de:tuda-tuprints-119154
URI: https://tuprints.ulb.tu-darmstadt.de/id/eprint/11915
Tag der mündlichen Prüfung: 26. Juni 2020

Abstract

Most enterprises and organizations have digitized their work by implementing process-aware information systems. These systems typically record a large amount of event data about the organization's day-to-day business. Organizations are interested in analyzing this data to optimize their business processes. Process mining is a research field that analyzes event data to obtain valuable knowledge about how processes are executed in reality. Different from conventional interview-based methods, process mining analyzes the process performance and compliance solely based on recorded event data.

Currently, the work of analysts using process mining tools is characterized as largely manual, leading to many ad-hoc tasks. With the growth of available event data and the increasing complexity of processes, several issues emerge in practice. Many process mining methods are not specifically designed for large event logs, leading to high computation time and a substantial amount of manual work. Consequently, suitable subsets of cases must be selected before applying certain analysis tasks successfully. Furthermore, applying process mining techniques to unknown event data often requires extensive domain knowledge and process mining expertise to obtain valuable insights. Although efforts were made to systematize the manual work of analysts in process mining projects, current tools lack intelligent computer-assisted guidance.

This dissertation introduces several contributions to different steps along an analyst's workflow to address the above issues. It is divided into three major parts: The first part introduces a process knowledge artifact framework that simplifies data extraction and processing of heterogeneous data sources as an enabling preparatory step towards process mining. In the second part, the dissertation introduces three algorithms specifically designed for large and complex real-life event logs. A novel compliance checking algorithm significantly improves runtime performance for large event logs to identify compliance rule violations. A parameter-free process drift detection algorithm automatically detects changes in the process execution over time to reveal potential process issues. A novel multi-perspective trace clustering algorithm automatically detects patterns in the control flow and the data perspective of a process to reveal the different process behaviors in event logs, aiming at simplifying process discovery. The third part of the dissertation integrates the contributions made in the first two parts and introduces an interactive visual recommendation approach to enhance process analysis guidance. Lastly, a

process improvement approach is presented, suggesting modifications to process models for a given improvement goal.

The proposed algorithms and methods were evaluated with synthetic and real-life event logs, demonstrating their superior qualitative performance and practical feasibility. A user study conducted with process mining experts evaluated the prototype implementation of the interactive visual recommendation system to assess the usefulness in the process analysis workflow.

Zusammenfassung

Viele Unternehmen haben ihre Arbeitsabläufe mittels prozessorientierter Informationssysteme digitalisiert. Diese Informationssysteme sammeln eine große Menge an Ereignisdaten über das Tagesgeschäft von Unternehmen. Durch die Analyse dieser Daten erhoffen sich Unternehmen ihre Geschäftsprozesse zu optimieren. Process Mining ist ein Forschungsgebiet, das solche Ereignisdaten analysiert, um wertvolle Erkenntnisse über den tatsächlichen Ablauf von Prozessen zu gewinnen. Im Gegensatz zu Interview-basierten Methoden nutzt Process Mining ausschließlich aufgezeichnete Ereignisdaten, um Prozessperformance und Konformitätsverstöße zu ermitteln.

Die Analyse von Geschäftsprozessen mittels aktueller Process-Mining-Werkzeuge ist jedoch zum größten Teil durch viele manuelle Arbeitsschritte charakterisiert. Das stetige Wachstum an verfügbaren Daten sowie die zunehmende Komplexität von Geschäftsprozessen führen in der Praxis daher zu erheblichen Problemen bei der Analyse. Viele Process-Mining-Methoden wurden nicht speziell für große Ereignisprotokolle optimiert, was oft zu langen Berechnungszeiten führt. Daher müssen oft sinnvolle Teilmengen selektiert werden, um die Analyse erfolgreich ausführen zu können. Zudem erfordern viele Process-Mining-Methoden umfangreiches prozessspezifisches Fachwissen sowie Process-Mining-Knowhow, um wertvolle Erkenntnisse zu gewinnen. Trotz der Systematisierung von Process-Mining-Aufgaben fehlen derzeitigen Werkzeugen intelligente computergestützte Analyseempfehlungen.

Diese Arbeit stellt mehrere Beiträge entlang der Vorgehensweise von Analysten im Process Mining vor, um die beschriebenen Problemfelder zu adressieren. Die Arbeit ist in drei Hauptteile gegliedert: Im ersten Teil wird ein Rahmenwerk vorgestellt, welches die Datenextraktion und -verarbeitung von Prozessdaten aus heterogenen Datenquellen vereinfacht, um Process Mining anwenden zu können. Der zweite Teil stellt drei Algorithmen vor, die speziell für die Verarbeitung von großen und komplexen Ereignisprotokollen entwickelt wurden. Ein neuartiger Compliance-Checking-Algorithmus erkennt Konformitätsverstöße auch in sehr umfangreichen Ereignisprotokollen und zeichnet sich dabei durch deutlich verbesserte Laufzeit aus. Ein parameter-freier Process-Drift-Detection-Algorithmus detektiert automatisiert Veränderungen des Prozessverhaltens über die Zeit in Ereignisprotokollen, die zu potenziellen Prozessproblemen führen können. Ein neuartiger Multi-Perspective-Trace-Clustering-Algorithmus extrahiert verschiedene Prozessverhalten in Ereignisprotokollen auf Basis von Kontrollfluss und Datenattributen automatisch, um Process Discovery zu verbessern. Der dritte Teil der

Arbeit integriert die ersten beiden Teile und stellt ein interaktives Analyseempfehlungssystem vor, das Analysten konkrete Analysevorschläge erzeugt. Schließlich wird ein Ansatz vorgestellt, der auf Basis zuvor definierter Ziele Verbesserungen an Prozessmodellen vorschlägt.

Die vorgestellten Algorithmen und Methoden wurden mittels synthetischer und realer Ereignisprotokollen ausgewertet, um deren qualitative Leistung und deren praktischen Nutzen zu demonstrieren. Mittels einer durchgeführten Nutzerstudie mit Process Mining Experten wurde das interaktive Analyseempfehlungssystem ausgewertet, um dessen Nützlichkeit im Rahmen von Process Mining Projekten zu bewerten.

Publications

Parts of this dissertation have been previously published in the following peer-reviewed publications:

[See+19a] Alexander Seeliger, Alejandro Sánchez Guinea, Timo Nolle, and Max Mühlhäuser. "ProcessExplorer: Intelligent Process Mining Guidance." In: *Business Process Management*. Cham: Springer International Publishing, 2019.

[SNM17] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. "Detecting Concept Drift in Processes using Graph Metrics on Process Graphs." In: *Proceedings of the 9th Conference on Subject-oriented Business Process Management - S-BPM ONE '17*. ACM Press, 2017. DOI: 10.1145/3040565.3040566.

[SNM18a] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. "Finding Structure in the Unstructured: Hybrid Feature Set Clustering for Process Discovery." In: *Business Process Management*. Cham: Springer International Publishing, 2018, pp. 288–304. DOI: 10.1007/978-3-319-98648-7_17.

[SNM18b] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. "Process Explorer: An Interactive Visual Recommendation System for Process Mining." In: *KDD Workshop on Interactive Data Exploration and Analytics* (2018).

[See+16a] Alexander Seeliger, Timo Nolle, Benedikt Schmidt, and Max Mühlhäuser. "Process Compliance Checking using Taint Flow Analysis." In: *Proceedings of the 37th International Conference on Information Systems - ICIS '16*. 2016.

[See+19b] Alexander Seeliger, Maximilian Ratzke, Timo Nolle, and Max Mühlhäuser. "ProcessExplorer: Interactive Visual Exploration of Event Logs with Analysis Guidance." In: *Proceedings of the 1st International Conference on Process Mining - ICPM'19 - Demo*. 2019.

[See+16b] Alexander Seeliger, Benedikt Schmidt, Immanuel Schweizer, and Max Mühlhäuser. "What Belongs Together Comes Together. Activity-centric Document Clustering for Information Work." In: *Proceedings of the 21st International Conference on Intelligent User Interfaces - IUI '16*. New York, NY, USA: ACM Press, 2016, pp. 60–70. DOI: 10.1145/2856767.2856777.

[SSM18] Alexander Seeliger, Michael Stein, and Max Mühlhäuser. “Can We Find Better Process Models? Process Model Improvement Using Motif-Based Graph Adaptation.” In: *Business Process Management Workshops*. Cham: Springer International Publishing, 2018, pp. 230–242. DOI: 10.1007/978-3-319-74030-0_17.

Acknowledgments

Without the wonderful support and encouragement from my advisors, colleagues, friends, and family, this thesis would not have been possible. Many thanks to each and every one of you!

First, I would like to thank Max Mühlhäuser (Technische Universität Darmstadt, Germany), my doctoral advisor and mentor, for his continuous support, encouragement, confidence, and advice over the last years which allowed me writing this thesis. I would also like to thank Michael Rosemann (Queensland University of Technology, Australia) for acting as a co-referee for my work.

A big thank you goes to all current and former colleagues of the Telecooperation Lab. It has been a great pleasure and fun to work with all of you in this fantastic research group over the past years. Especially, I would like to thank everyone from the Intelligent Systems (ISY) group for all the support and your contributions to my work: Timo, Christian, Sebastian, and Alejandro. I will not forget the numerous hours of discussion, which produced exciting thoughts and ideas. You are all awesome!

Furthermore, I also would like to thank Benedikt who introduced me to the field of process mining after my master thesis. Also, a big thank you to Tobias from PAF (Process Analytics Factory), whose passion for process mining encouraged me conducting research in this area and who supported me during this journey.

Finally, I would like to thank my parents, Edith and Wolfgang, as well as my friends. Without all your support, I would not have been able to complete this thesis.

Thank you.

Contents

I **Introduction**

1 **Introduction** 3
- 1.1 Problems and Research Questions . . . 4
- 1.2 Contributions . . . 7
- 1.3 Outline . . . 8

2 **Background** 11
- 2.1 Basic Notations . . . 11
- 2.2 Event Logs . . . 12
  - 2.2.1 Event Log Notation . . . 12
  - 2.2.2 Simple Event Log Notation . . . 15
  - 2.2.3 Process Perspectives . . . 15
- 2.3 Process Mining . . . 16
  - 2.3.1 Process Discovery . . . 17
  - 2.3.2 Conformance Checking . . . 18
  - 2.3.3 Enhancement . . . 19
- 2.4 Process Discovery . . . 19
  - 2.4.1 Process Map . . . 21
  - 2.4.2 Flexible Heuristics Miner (FHM) . . . 22
  - 2.4.3 Process Model Quality Dimensions . . . 25
- 2.5 Data Mining . . . 28
  - 2.5.1 Cluster Analysis . . . 28
  - 2.5.2 Frequent Pattern Mining . . . 33
- 2.6 Summary . . . 36

II **Knowledge Extraction**

3 **Process Knowledge Artifacts** 41
- 3.1 Introduction and Motivation . . . 41
- 3.2 Related Work . . . 43
  - 3.2.1 Data Collection and Consolidation . . . 43
  - 3.2.2 Process Analysis Workflow . . . 44
- 3.3 Process Mining Tasks Analysis . . . 45
  - 3.3.1 Methodology . . . 45
  - 3.3.2 Results . . . 46
- 3.4 Framework . . . 49
  - 3.4.1 Data Collection and Consolidation Layer . . . 50
  - 3.4.2 Process Analysis Workflow Layer . . . 54
- 3.5 Case Study . . . 58

3.5.1 Experiment Setup . . . 59
3.5.2 Reduction in Manual Work . . . 59
3.5.3 Knowledge Modeling . . . 60
3.5.4 Comparison with the State of the Art . . . 61
3.6 Discussion and Limitations . . . 63
3.6.1 Meta-Model . . . 64
3.6.2 Targeted Analysis . . . 64
3.6.3 Case Study . . . 64
3.7 Conclusion . . . 64

III **Process Analysis**

4 **Business Rule Evaluation using TFA** 69
4.1 Introduction and Motivation . . . 69
4.2 Related Work . . . 72
4.2.1 Process Model-based Compliance Checking . . . 72
4.2.2 Rule-based Compliance Checking . . . 74
4.3 Compliance Checking as a Reachability Problem . . . 75
4.3.1 Taint Flow Analysis . . . 76
4.3.2 Distributive Functions . . . 81
4.3.3 Tabulation Algorithm . . . 86
4.3.4 Replay of Process Variants on Exploded Supergraph . . . 87
4.4 Evaluation . . . 88
4.4.1 Experiment Setup . . . 88
4.4.2 Measurement Method . . . 91
4.4.3 Results . . . 92
4.5 Discussion and Limitations . . . 96
4.5.1 Compliance Rules . . . 96
4.5.2 Process Map . . . 96
4.5.3 Counter-example . . . 96
4.6 Conclusion . . . 97

5 **Process Drift Detection** 99
5.1 Introduction and Motivation . . . 100
5.2 Related Work . . . 102
5.2.1 Adaptive Process Management Systems . . . 103
5.2.2 Similarity-based Detection . . . 104
5.2.3 Statistical Hypothesis Testing based Detection . . . 105
5.3 Detecting Process Drifts using Graph Metrics . . . 106
5.3.1 Splitting Event Log into Reference and Detection Window 106
5.3.2 Compute Process Models and Graph Metrics . . . 108
5.3.3 Perform a G-Test on the Graph-Metrics . . . 110
5.3.4 Characterization of the Process Drifts . . . 111
5.4 Evaluation . . . 113
5.4.1 Experiment Setup . . . 113
5.4.2 Accuracy Results . . . 116

5.4.3 Characterization Extraction Results . . . 118
5.5 Discussion and Limitations . . . 121
5.5.1 Drift Types . . . 121
5.5.2 Process Model Discovery . . . 121
5.5.3 Visualization . . . 122
5.6 Conclusion . . . 122

6 **Hybrid Feature Set Trace Clustering** 125
6.1 Introduction and Motivation . . . 126
6.2 Related Work . . . 128
6.2.1 Similarity-based Trace Clustering . . . 128
6.2.2 Model-based Trace Clustering . . . 131
6.3 Hybrid Feature Set Clustering . . . 132
6.3.1 Generation of Candidate Clusters . . . 133
6.3.2 Generation of Non-Overlapping Clusters . . . 136
6.3.3 Automatic Parameter Tuning . . . 138
6.4 Evaluation . . . 140
6.4.1 Synthetic Event Logs Evaluation . . . 140
6.4.2 Real-life Event Logs Evaluation . . . 146
6.5 Discussion and Limitations . . . 150
6.5.1 Feature Space . . . 151
6.5.2 Interestingness . . . 151
6.5.3 Runtime Performance . . . 152
6.6 Conclusion . . . 152

IV **Process Analysis Assistance**

7 **Intelligent Browsing** 157
7.1 Introduction and Motivation . . . 158
7.2 Related Work . . . 159
7.2.1 Exploratory Data Analysis . . . 159
7.2.2 Data Insights . . . 160
7.2.3 Interactive Browsing in Process Mining . . . 161
7.3 ProcessExplorer Approach . . . 163
7.3.1 Discovery of Subset Recommendations . . . 163
7.3.2 Discovery of Insight Recommendations . . . 164
7.3.3 Ranking of Recommendations . . . 167
7.4 ProcessExplorer System . . . 167
7.4.1 User Interface . . . 167
7.4.2 Architecture . . . 173
7.5 Evaluation . . . 175
7.5.1 User Study: Identify Key Requirements for Analytic Guidance . . . 175
7.5.2 User Study: Evaluation of Usefulness . . . 178
7.6 Discussion and Limitations . . . 182
7.6.1 Recommendations . . . 182

7.6.2 Usefulness . . . 182
7.6.3 Runtime Performance . . . 183
7.7 Conclusion . . . 183

8 **Process Improvement** 185
8.1 Introduction and Motivation . . . 186
8.2 Related Work . . . 188
8.3 Finding Appropriate Target Signature . . . 189
8.4 Motif-based Process Adaptation . . . 191
8.4.1 Process Map Adaptation . . . 191
8.4.2 Specification of Process Constraints . . . 195
8.5 Evaluation . . . 196
8.5.1 Experiment Setup . . . 196
8.5.2 Results . . . 197
8.6 Discussion and Limitations . . . 200
8.6.1 Improvement Goal . . . 200
8.6.2 Local Improvements . . . 201
8.6.3 Process Model . . . 202
8.7 Conclusion . . . 202

v **Closure**

9 **Summary and Conclusion** 207
9.1 Summary . . . 207
9.2 Directions for Future Research . . . 210

vi **Appendix**

A **Business Rule Evaluation using TFA** 213

B **Process Drift Detection** 215

C **Hybrid Feature Set Trace Clustering** 219

Bibliography 223

List of Figures

Figure 1.1 Overview of the PM^2 methodology. Adapted from [Eck+15]. . . . 4
Figure 1.2 Overview of the three main research questions and six contributions introduced in this thesis. . . . 5
Figure 1.3 Structure of this thesis. . . . 10
Figure 2.1 A simple event log obtained by transforming the three traces of the example event log shown in Table 2.1. . . . 15
Figure 2.2 Overview of the three tasks of process mining [Aal11]. 17
Figure 2.3 Directly-follows graph of a small sample of the BPI Challenge 2019 event log. . . . 18
Figure 2.4 Process model obtained with the Inductive Visual Miner [LFA14a] of a small sample of the BPI Challenge 2019 event log. . . . 18
Figure 2.5 Conformance checking result of the Inductive Visual Miner for a small sample of the BPI Challenge 2019 event log. . . . 19
Figure 2.6 Performance overview on top of a process tree obtained with the Inductive Visual Miner from a small sample of the BPI Challenge 2019 event log. . . . 19
Figure 2.7 C-net derived from the event log L. . . . 25
Figure 2.8 Venn diagram illustrating the relations between the behavior of the process model, the observed behavior in the event log, and system's behavior. Adapted from [BDA14]. . . . 26
Figure 2.9 Illustration of the k-means algorithm. . . . 29
Figure 2.10 Illustration of the two hierarchical clustering methods. 31
Figure 2.11 FP-close tree for a transaction database with the corresponding header table. Adapted from [HKP11] 36
Figure 3.1 Schematic structure of the generation of an event log from raw data sources using business objects. . . . 55
Figure 3.2 Overview of time spent in process mining projects comparing the process knowledge artifact framework and conventional process mining tools. Interview-based consulting results were obtained from former experience of the consulting company. . . . 59
Figure 3.3 Overview of the chapter and contributions. . . . 65

Figure 4.1 Overview of the steps performed during the compliance checking using taint flow analysis. 76
Figure 4.2 Path example for a given process map of a simple procurement process [See+16a]. 79
Figure 4.3 Example paths for producer, sanitizer, and other activities represented as a graph. 83
Figure 4.4 An example procurement handling process represented as a process map and the corresponding exploded supergraph. 84
Figure 4.5 Causal net discovered with the heuristics miner of ProM showing the event log A. 90
Figure 4.6 Average runtime of the compliance checking approaches over a total number of 5 runs. 93
Figure 4.7 Processing time in seconds for the synthetic event logs depending on the number of traces in the log. 94
Figure 4.8 Processing time in seconds for the synthetic event logs depending on the number of rules in the log. 95
Figure 4.9 Overview of the chapter and contributions. 97
Figure 5.1 Overview of the different process drift types. 101
Figure 5.2 Overview of the Process Drift Detection Algorithm. . . 107
Figure 5.3 Process model in Business Process Model Notation (BPMN) with a process drift marked in red [SNM17]. . 111
Figure 5.4 Resulting annotations to the process model in BPMN with a process drift marked in red. 112
Figure 5.5 Base BPMN model of the benchmark dataset [Maa+15]. 114
Figure 5.6 F1-score across all process drift patterns compared with the related work. 116
Figure 5.7 Average delay across all process drift patterns. 117
Figure 5.8 Process model of the *cb* process drift pattern event log [Maa+15]. 119
Figure 5.9 Overview of the chapter and contributions. 123
Figure 6.1 Process map of the BPI Challenge 2019 event log showing all observed activities and transitions. 126
Figure 6.2 Overview of trace clustering applied to a single event log. 127
Figure 6.3 Architectural overview of the hybrid feature set trace clustering algorithm. 132
Figure 6.4 The Levenshtein distance matrix which shows the calculation of the costs and the minimum cost path. . . . 134
Figure 6.5 Causal net discovered by the heuristics miner of ProM showing the synthetic P2P event log. 141
Figure 6.6 The synthetic event log generation containing different process behavior that considers the control flow and case attributes. 142

Figure 6.7 Process model fitness after trace clustering aggregated across all synthetic event logs with a noise level of 0.2. 144
Figure 6.8 Process model precision after trace clustering aggregated across all synthetic event logs with a noise level of 0.2. 145
Figure 6.9 F1-BCubed results after trace clustering aggregated across all synthetic event logs with a noise level of 0.2. 145
Figure 6.10 Adjusted rand index results after trace clustering aggregated across all synthetic event logs with a noise level of 0.2. 146
Figure 6.11 Critical difference diagram for all methods on all synthetic event logs. 147
Figure 6.12 Overview of the chapter and contributions. 153
Figure 7.1 User interface of Fluxicon Disco showing the discovered process model of the BPI Challenge 2019 event log. 162
Figure 7.2 User interface of PAFnow Process Mining showing the process step evaluation tab of the BPI Challenge 2019 event log. 162
Figure 7.3 User interface of ProcessExplorer. 168
Figure 7.4 Subset recommendations of the BPI Challenge 2019 event log. 169
Figure 7.5 Statistics of a selected subset recommendation of the BPI Challenge 2019 event log. 170
Figure 7.6 Insight recommendations of a selected subset recommendation of the BPI Challenge 2019 event log. 171
Figure 7.7 Process map of ProcessExplorer showing a subset recommendation of the BPI Challenge 2019 event log. . 172
Figure 7.8 Stage view navigation of ProcessExplorer. 172
Figure 7.9 Architectural overview of the ProcessExplorer system [See+19b]. 174
Figure 7.10 User interface of the early prototype of ProcessExplorer. 176
Figure 7.11 Results of the User Experience Questionnaire (UEQ) for the requirements study and for the final prototype. 178
Figure 7.12 Overview of the TAM usefulness estimation for ProcessExplorer according to the question clusters. 180
Figure 7.13 Overview of the chapter and contributions. 184
Figure 8.1 Overview of all directed motifs with 3 nodes 186
Figure 8.2 Overview of the business process motif-based graph adaptation approach [SSM18]. 187
Figure 8.3 3 node motif frequency over the investigated process models with two different heuristic miner settings. . . 190
Figure 8.4 Pearson correlations between motif and graph property.190

Figure 8.5 Cluster coefficient and density values before and after the optimization grouped for the three different target signatures. 198
Figure 8.6 Distance to target signatures over the number of iterations. 200
Figure 8.7 Normalized number of edge operations performed for investigated target signatures and iterations. 201
Figure 8.8 Overview of the chapter and contributions. 203
Figure A.1 Processing time in seconds for the synthetic event logs depending on the number of rules in the log (Part 1). . 213
Figure A.2 Processing time in seconds for the synthetic event logs depending on the number of rules in the log (Part 2). . 214
Figure B.1 F1-score for different process drift patterns (higher is better) and compared with the related work. 216
Figure B.2 Average delay for different process drift patterns. . . . 217
Figure C.1 Process model fitness after trace clustering aggregated over all synthetic event logs depending on the noise level. 219
Figure C.2 Process model precision after trace clustering aggregated over all synthetic event logs depending on the noise level. 220
Figure C.3 F1-BCubed results after trace clustering aggregated over all synthetic event logs depending on the noise level. 221
Figure C.4 Adjusted rand index results after trace clustering aggregated over all synthetic event logs depending on the noise level. 222

List of Tables

Table 2.1 Simplified example event log of a procurement handling process. . . . 13
Table 2.2 Frequency of the directly-follows relation $|a >_L b|$ of the event log L. . . . 24
Table 2.3 Dependency measure between the activities of the event log L. . . . 24
Table 3.1 Tasks performed during process analysis grouped by BPI Challenge. . . . 47
Table 3.2 Excerpt of the EKKO table of SAP MM. . . . 52
Table 3.3 Excerpt of the EKPO table of SAP MM. . . . 52
Table 3.4 Excerpt of the EKBE table of SAP MM. . . . 52
Table 3.5 Resulting event log table of the Purchase Order business object. . . . 53
Table 3.6 Resulting case attribute table of the Purchase Order business object. . . . 53
Table 3.7 Targeted analysis tasks that can be used in a Process Knowledge Artifact (PKA). . . . 56
Table 3.8 Simplified example PKA investigating in cases with duration times between *PO_created* and *Goods_receipt* longer than 30 days. . . . 57
Table 3.9 Overview of the process mining projects which were used to evaluate the process knowledge artifact framework. . . . 59
Table 3.10 Overview of the different process mining tools compared with the process knowledge artifact framework. 62
Table 4.1 Property comparison of process compliance checking algorithms. . . . 73
Table 4.2 General properties of the event logs used in the evaluation. . . . 89
Table 4.3 Compute server specifications. . . . 91
Table 4.4 Number of cases that violate against a compliance check. 92
Table 4.5 Performance Benchmark (in milliseconds) for Taint Flow Analysis, LTL Checker, and Petri net Replay. . . . 94
Table 5.1 Property comparison of process drift detection algorithms. . . . 103
Table 5.2 Graph metrics computed by the process drift detection algorithm. . . . 112

Table 5.3 Process drift patterns of the benchmark event logs [WRR08; Maa+15]. 115
Table 5.4 Evaluation setup of all compared process drift detection methods. 115
Table 5.5 Results of the experimental evaluation of the process drift detection methods. 117
Table 5.6 Observed graph metric changes and absolute values of transitions for the given example. 118
Table 5.7 Characterizations extracted from the contributed algorithm for each process drift pattern. 120
Table 6.1 Property comparison of trace clustering algorithms. . . 129
Table 6.2 Process models used for generating the event log: Number of activity types (# at), number of transitions (# tr), number of variants (# dpi), maximal trace length, and out-degree. 141
Table 6.3 Performance of the related work and the hybrid cluster approach with respect to process model and clustering evaluation. 147
Table 6.4 Overview of the real-life event logs investigated. 148
Table 6.5 Results of the real-life event logs showing the average weighted fitness. 149
Table 6.6 Results of the real-life event logs showing the average weighted precision. 149
Table 6.7 Results of the real-life event logs showing the number of clusters. 150
Table 6.8 Results of the real-life event logs showing the average weighted simplicity. 151
Table 7.1 Case- and subset-based process performance indicators (PPIs). 165
Table 7.2 The questions and results of the Technology Acceptance Model (TAM) usefulness estimation. 179
Table 8.1 Semantics for the DECLARE constraints [Bur+12]. . . . 195
Table 8.2 Symbols for the LTL expressions. 195
Table 8.3 Characteristics of the used real-life event logs and the number of extracted process constraints. 197
Table 8.4 Evaluation result of the process improvement approach for the standard Flexible Heuristics Miner setting. . . . 199
Table 8.5 Evaluation result of the process improvement approach for the noise-free Flexible Heuristics Miner setting. . . 199
Table B.1 Detailed results of the experimental evaluation of the process drift detection methods. 215

Acronyms

ATC	ActiTraC
ARI	Adjusted Rand Index
BI	Business Intelligence
BOA	Bag-of-Activities
BPIC	Business Process Intelligence Challenge
BPMN	Business Process Model Notation
CAC	Context-Aware-Clustering
DG	Dependency Graph
ETL	Extract Transform Load
EPC	Event-driven Process Chain
FHM	Flexible Heuristics Miner
GDPR	General Data Protection Regulation
HC	Hybrid Feature Set Trace Clustering
LED	Levenshtein Edit Distance
LTL	Linear Temporal Logic
LoMbA	Local Motif-based Adaptation
PAIS	Process-aware Information System
RI	Rand Index
PSO	Particle Swarm Optimization
PKA	Process Knowledge Artifact
PPI	Process Performance Indicator
TAM	Technology Acceptance Model
UEQ	User Experience Questionnaire

Part I

Introduction

1 Introduction

In recent years, more and more mid and large size enterprises started to collect event data about their daily business with the goal of optimizing and improving their work. This new practice usually leads to huge datasets that must be processed automatically. More concretely speaking, large investments have been made towards the digitization of information systems that coordinate and support business processes across the organization. Analyzing such process-related data offers organizations new opportunities to achieve more efficient and effective process execution. However, the increasing process complexity and the automation of essential tasks in production or services make it difficult to obtain a holistic view of a specific process. Tasks such as business process improvement, optimization, and process automation become increasingly challenging without having a deep understanding of the actual behavior of the process.

Process mining analyzes event data of Process-aware Information Systems (PAISs) to obtain a better understanding of how processes are executed in reality [Aal11]. It provides valuable insights to improve process execution, predict process outcomes and identify potential compliance issues. Process mining is categorized into three major tasks: process discovery, conformance checking, and process enhancement [Aal11]. *Process discovery* reconstructs an as-is process model solely based on the recorded event data. *Conformance checking* checks if the actual process behavior observed in reality conforms to an existing process model and vice versa. *Process enhancement* extends and improves existing process models with information from event data to better reflect reality. Process mining techniques have been successfully applied to event data of different processes and organizations. The results have been documented in many case studies [Aal+07; Man+08; Goe+11; LL14; Eck+15; Váz+16; CJ17; Man18]. During the last decade, various open-source and commercial tools have emerged. The process mining discipline is growing rapidly, and these tools are making it accessible to a wider range.

Due to the growth of available event data and the increase of process complexity, it is increasingly challenging to gain interesting insights with the help of process mining techniques [Eck+15; Nie15]. Analysts require extensive domain knowledge [Aal+07; Goe+11] and process mining expertise [LL14] to obtain the desired results. In practice, analysts still carry out most of their process mining activities manually. As a consequence, process mining projects

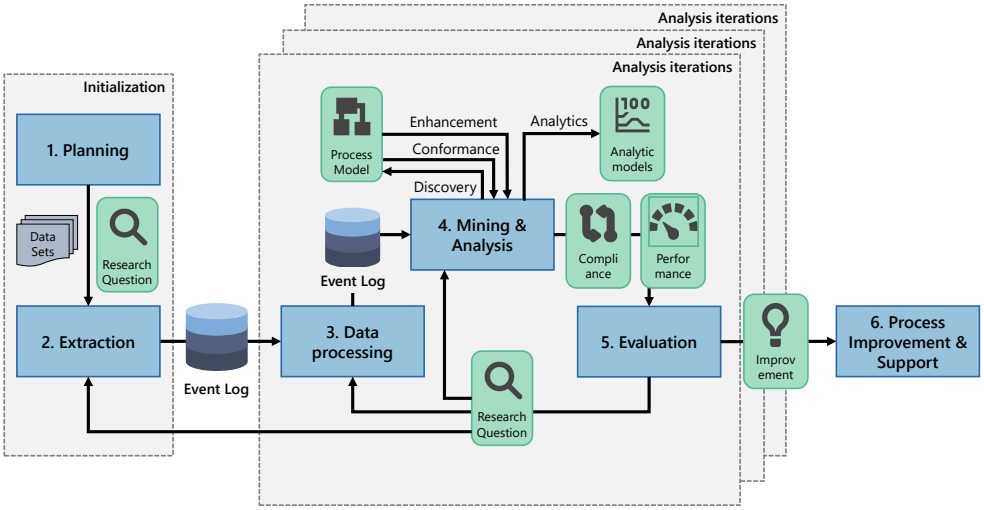


Figure 1.1: Overview of the PM^2 methodology. Adapted from [Eck+15].

are typically supported by external consultants. Various efforts have been made to systematize process mining in order to guide the planning and execution of process mining projects and to reduce costs and time [BGW09; Aal11; Eck+15]. One of such efforts is PM^2 [Eck+15], which is a well-established methodology that describes in six stages how to conduct process mining projects successfully. Figure 1.1 illustrates the stages of the methodology and how they are interconnected with each other.

Despite the increasing number of available tools which incorporate process mining and machine learning capabilities, the existing tools still lack intelligent computer-assisted support to reduce manual work. This lack of support results in repetitive and time-consuming tasks hampering the efficient analysis of processes. Ad-hoc work is needed to: (i) prepare event logs; (ii) scan through the massive amount of data; (iii) interpret highly complex process models; and (iv) compute process performance measures to receive the desired output.

This thesis aims to address the above issues by providing several intelligent computer-assisted methods that improve the support for process mining analysts working with event logs. In particular, we propose enhancements to the state of the art in three different areas of process mining: knowledge extraction, process analysis, and process analysis assistance.

1.1 Problems and Research Questions

The research work presented in this thesis addresses three main research questions and challenges in the practical application of process mining techniques. Our work is split into different areas that determine the three main parts of this thesis alongside the introduction and conclusion part: knowledge

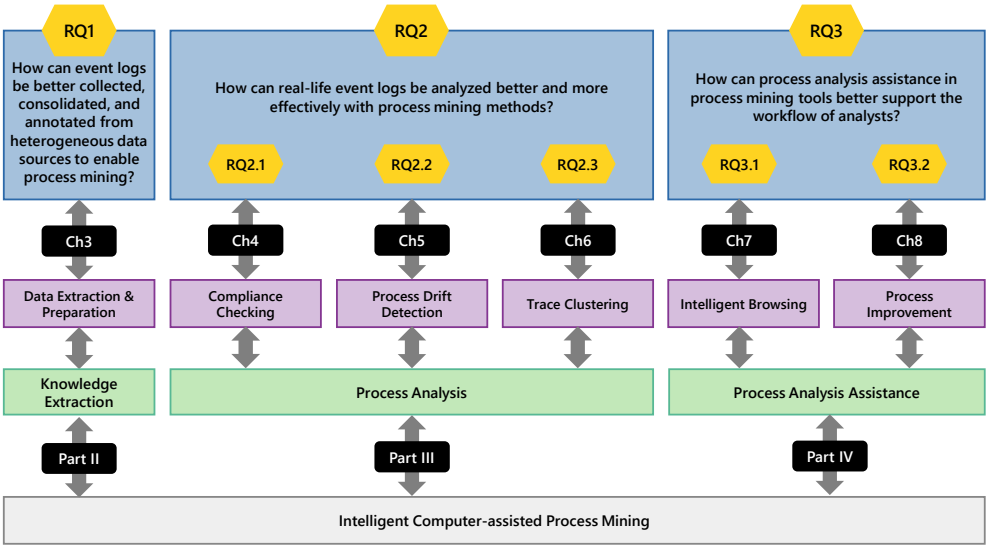


Figure 1.2: Overview of the three main research questions and six contributions introduced in this thesis.

extraction, process analysis, and process analysis assistance. These three main areas of this thesis are linked to the different stages of the PM^2 methodology. Figure 1.2 depicts the links between the main research questions and contributions.

Knowledge Extraction

The first part of this thesis investigates the first three stages of the PM^2 methodology that deal with "(1) Planning", "(2) Extraction", and "(3) Data processing". This part of the thesis aims at answering the following research question:

RQ1: *How can event logs be better collected, consolidated, and annotated from heterogeneous data sources to enable process mining?*

The basis for almost any process mining technique is an event log which stores information about the actual execution of a process in information systems. However, event data is rarely explicitly stored in a consistent process event database but is distributed in different data sources such as plain text files or relational databases. This data needs to be collected and then transformed into event logs for use in process mining [IEE11]. According to previous case studies, data preparation can consume up to 90% of the total process mining project time [LL14]. In addition, data entities often need to be translated into human-readable entities because they are usually stored technically. Incomplete process data and the lack of domain knowledge in event logs lead to incorrect assumptions and interpretations. Consequently, process mining results are misleading or provide incorrect findings [Eck+15].

Process Analysis

The second part of this thesis deals with the "(4) Mining & Analysis" stage of the process mining methodology (see Figure 1.1). This part of the thesis aims to answer the following main research question:

RQ2: *How can real-life event logs be analyzed better and more effectively with process mining methods?*

Many process mining methods cannot easily be applied to real-life event logs. The increase of available event data and the flexibility of processes lead to several issues with the current methods. We divide our main research question for this part into three more specific research questions to address these issues:

- **RQ2.1:** *How can compliance rules be efficiently evaluated in large real-life event logs?*
- **RQ2.2:** *How can process drifts be detected in real-life event logs without any prior knowledge about the process?*
- **RQ2.3:** *How can the different process behaviors, in flexible environments, be extracted from real-life event logs without any prior knowledge about the process?*

An important area of interest for organizations is compliance checking. However, compliance checking of large event logs leads to high computing times, making it unusable for event logs of industrial size [MCA13]. Another problem area of current process mining methods is the assumption of a single process behavior within an event log. Practice has shown that flexible environments lead to heterogeneous event logs with different process behaviors [SGA09; ETF16; Wee+13]. These different behaviors are often not considered when applying process mining techniques.

Process Analysis Assistance

In the third part of this thesis, the last two stages of the PM^2 methodology are investigated. The stages "(5) Evaluation" and "(6) Process Improvement & Support" interpret the results of the process analysis and provide actions for improving processes. This part of the thesis aims to answer the following research question:

RQ3: *How can process analysis assistance in process mining tools better support the workflow of analysts?*

Currently, the workflow of analysts using process mining in practice is characterized largely as a set of manual tasks. Guiding analysts to interesting

and valuable insights can help to further improve processes. In this respect, we aim to answer the following two sub-research questions:

- **RQ3.1:** *How to design assistance for process mining tools to support the work of analysts?*
- **RQ3.2:** *How can insights of process mining analysis result in process model improvements?*

Many techniques require analysts to perform ad-hoc tasks, scan through massive amounts of data, apply other analysis techniques before visually inspecting the results (in order to discover valuable insights such as anomalies, inefficiencies, bottlenecks, or compliance violations). Typically, real-life event logs generate highly complex process models, which are called spaghetti models because of their appearance and pose a particular challenge for visual inspection. Despite the increasing number of process mining tools and established process mining methodologies, existing tools provide only limited intelligent computer-assisted process analysis guidance [IEE11; Eck+15; Nie15].

1.2 Contributions

This thesis contributes to the three areas introduced in the previous section, providing novel approaches that respond to the research questions raised above. This section briefly summarizes the contributions of this thesis per chapter. Figure 1.3 depicts the overall structure of the thesis.

Knowledge Extraction. Part II of the thesis introduces the following knowledge extraction method:

- A framework (Chapter 3) simplifies the workflow of analysts during the collection, preparation, and analysis of event data in process mining projects. This framework addresses RQ1 by introducing process knowledge artifacts. First, these artifacts close the gap between raw data stored in PAISs and the event log needed for process mining. Second, analysts can capture process analytical knowledge, i. e., the tasks needed to investigate a particular process problem, in a meta-model.

Process Analysis. In the process analysis part (Part III), RQ2 is addressed by means of three contributions, which each answer the subquestions of RQ2:

- A compliance rule evaluation method addresses RQ2.1 (Chapter 4). We formulate compliance checking as a graph reachability problem. The method returns a counterexample that can be visually inspected. It is specifically designed to work with very large event logs. The approach is applicable in various real-life scenarios.

- A parameter-free process drift detection method addresses RQ2.2 (Chapter 5). Our method uses graph measures of discovered process models to reliably detect changes to the process execution over time. Detected process drifts are characterized by returning the changes to the inspected graph measures to provide further insights about the identified process change.
- A parameter-free multi-perspective trace clustering method addresses RQ2.3 (Chapter 6). We combine the control flow and the data perspective of the process. The method searches for sets of process instances with similar behavior. The clustering result is automatically optimized with respect to process model fitness.

Process Analysis Assistance. Lastly, the following two contributions support analysts during their process analysis and, therefore, address RQ3:

- An interactive process mining browsing tool addresses RQ3.1 (Chapter 7). Our method provides exploratory suggestions during the analysis of large event logs. The tool recommends interesting subsets in the form of a case filter to guide the user through the event log exploration. Each subset is characterized by a set of automatically calculated analysis insights.
- A motif-based process improvement method addresses RQ3.2 (Chapter 8). We adapt a process model towards a given optimization goal. The method uses graph motifs to suggest potential process model modifications while retaining the feasibility of the process using constraints extracted from event logs.

The core contributions of this thesis were published in five peer-reviewed publications [See+16a; See+16b; SNM17; SNM18a; See+19a] at the International Conference on User Interfaces (IUI), the International Conference on Subject-oriented Business Process Management (S-BPM), the International Conference on Business Process Management (BPM), and the International Conference on Information Systems (ICIS). Additionally, two peer-reviewed workshop papers were published [SSM18; SNM18b] at the KDD Workshop on Interactive Data Exploration and Analytics (IDEA) and the Business Process Intelligence Workshop (BPI). The interactive process mining browsing tool was also presented as a demo [See+19b] at the the International Conference on Process Mining (ICPM).

1.3 Outline

This thesis is structured into five parts and 9 chapters. An overview of the structure is depicted in Figure 1.3.

PART I: INTRODUCTION The introductory Chapter 1 describes the context in which the contributions of this thesis are placed. Chapter 2 introduces the necessary background knowledge such as basic mathematical notations, expressions, data and process mining algorithms, and the notation for event logs.

PART II: KNOWLEDGE EXTRACTION Chapter 3 presents the process knowledge artifact framework, which allows analysts to model process analysis knowledge for reusable analysis workflows.

PART III: PROCESS ANALYSIS Chapter 4 introduces the use of taint flow analysis for evaluating business compliance rules. The algorithm is specifically designed to work with large real-life event logs. Chapter 5 introduces a parameter-free process drift detection algorithm based on graph metrics on top of discovered process models. Chapter 6 contributes a multi-perspective trace clustering approach that uses control flows and case attributes to generate subsets of similar cases.

PART IV: PROCESS ANALYSIS ASSISTANCE Chapter 7 introduces PROCESS-EXPLORER, an approach to enhance the interactive visual exploration of large real-life event logs. Chapter 8 presents a process improvement approach that suggests modifications to existing process models without affecting feasibility.

PART V: CONCLUSION Chapter 9 summarizes the thesis and highlights the contributions. Furthermore, this chapter gives an outlook on future work.

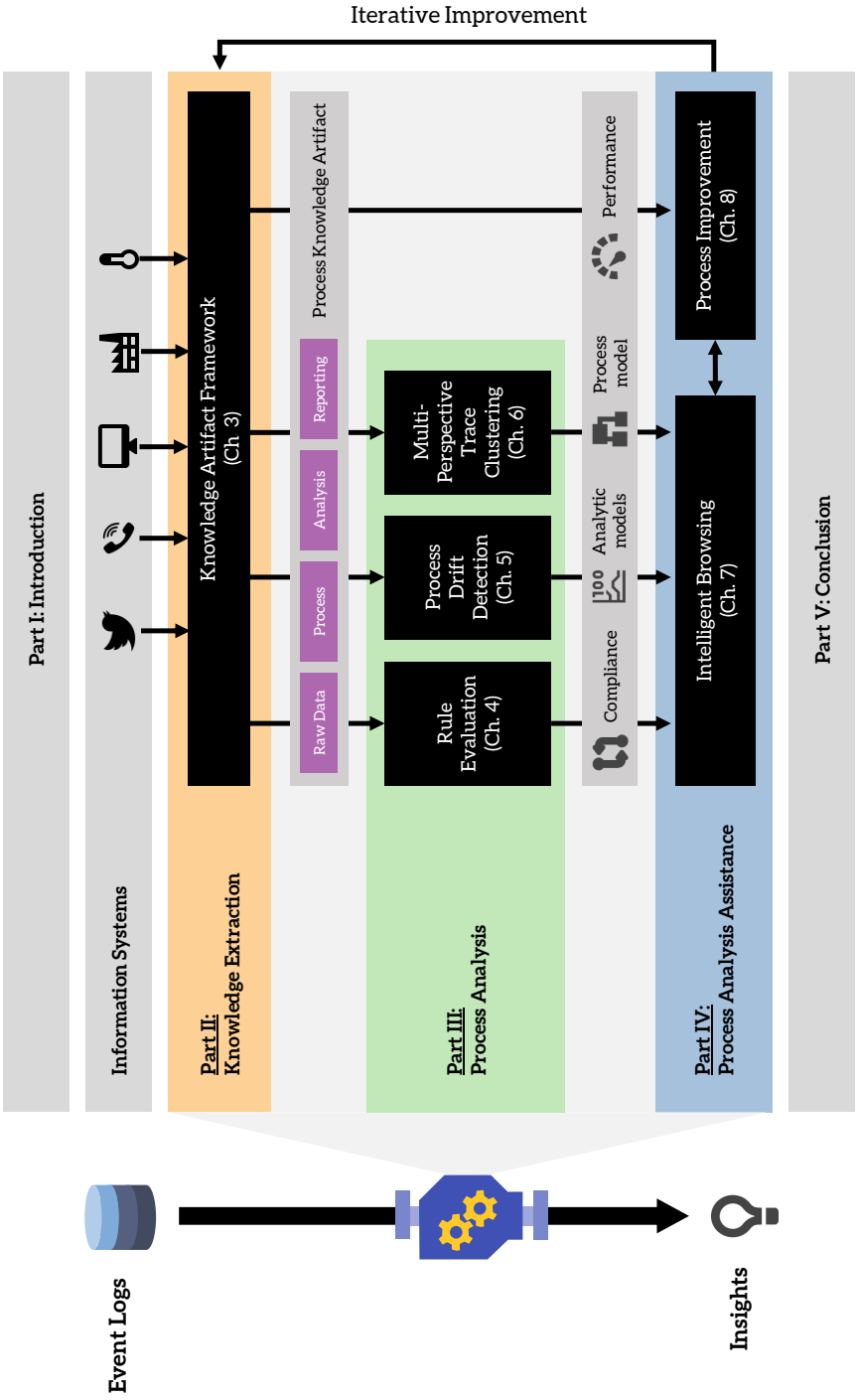


Figure 1.3: Structure of this thesis.

2

Background

In this chapter, the necessary background knowledge such as basic mathematical notations, expressions, data and process mining algorithms, and the notation for event logs are introduced.

2.1 Basic Notations

In this thesis, we use the usual mathematical notations for elements, sets, and logic operators. We use $X = \{a, b, c\}, \in, \notin, \subseteq, \subset, \cup, \cap, \setminus, \mathcal{P}(X), |X|$ for denoting *sets*, *element of*, *not element of*, *subset of*, *proper subset of*, *union*, *intersection*, *complement*, *power set*, and *cardinality*. Furthermore, we use $\mathbb{N} = \{1, 2, 3, ...\}$ to refer to the positive natural numbers.

Besides sets, the concept of *multi-sets* is used which allows the same object to be part of a set multiple times.

Definition 2.1 (Multi-set)**.** A multi-set is a tuple $M = (X, m)$, where X is the underlying set and $m : X \to \mathbb{N}$ is the multiplicity function. It indicates the number of occurrences of an element $x \in X$ in the multi-set M. Similar to sets, $x \in M$ denotes the containment of x in the multi-set, i. e., $m(x) \geq 1$.

This thesis uses a simplified notation of a multi-set, according to [Aal11]: $M = [a, a, b, b, b] = [a^2, b^3]$ denotes the multi-set $M = (\{a, b\}, m)$ with $m(a) = 2$ and $m(b) = 3$. Further, we denote $\mathbb{B}(D) = D \to \mathbb{N}$ as the set of multi-sets over a finite domain D, i. e., $X \in \mathbb{B}(D)$ is a multi-set, where $X(d)$ denotes the number of times d occurs in the multi-set for each $d \in D$.

Sequences of events are the basis for process mining and widely used in this thesis. A sequence is an ordered collection of objects in which objects can occur multiple times.

Definition 2.2 (Sequence)**.** Let A be a set of objects. A non-empty sequence over A is a function $\sigma : \{1, ..., n\} \to A$ with a sequence length of n. σ is the function that defines in which order objects of A appear in the sequence. A sequence is denoted as $\sigma = \langle a_1, ..., a_n \rangle$ where $a_i = \sigma(i)$, for $1 \leq i \leq n$. $a_i \in \sigma$ is the i-th object in the sequence. The empty sequence of length 0 is denoted as $\langle\rangle$.

We further denote A^* as the set of sequences over elements in A, i. e., for any $n \in \mathbb{N}$ and $a_1, a_2, ..., a_n \in A$: $\langle a_1, a_2, ..., a_n \rangle \in A^*$.

2.2 Event Logs

Process-aware Information Systems (PAISs) [Aal11], such as those offered by SAP, Oracle, or Microsoft, usually record event data, providing information about what activity was executed when, where, and by whom. Event data is often stored in *event logs*.

Table 2.1 shows an example event log of a procurement handling process recorded by a PAISs. We assume that each event log only contains information about a *single process*, i. e., a series of structured activities in order to achieve a particular business goal. Additionally, we assume that each event can be referred to a *single process instance*, also called *case*. In Table 2.1 each event is associated to a specific case, e. g., case 1 or case 2. We further assume that each event can be associated with an *activity*. For example, in Table 2.1 events are associated to activities like *Purchase request created*, *Purchase request release*, and *Purchase order created*. Lastly, we assume that events within a case are ordered, e. g., by timestamp.

In Table 2.1 cases and events have additional information, i. e., event attributes and case attributes. For example, a *timestamp* and a *resource*, e. g., the person that executed the event, is associated with each event. Each case is associated with the *vendor* and *category* attribute.

2.2.1 Event Log Notation

In the following, we give a formal definition of events and attributes.

Definition 2.3 (Event, Attribute [Aal16])**.** Let $\mathcal{E}$ be the set of all possible event identifiers that refer to a specific event. Events may be described by *attributes*, such as a timestamp. Let $\mathcal{A}$ be the set of attributes and $\mathcal{V}_a$ the set of all possible values of attribute $a \in \mathcal{A}$. For an event $e \in \mathcal{E}$ and an attribute $a \in \mathcal{A}$, let $\#_a(e)$ be the function that returns the value of attribute a for the event e.

In this thesis, we assume the existence of at least the following event attributes [Aal11]:

- $\#_{activity}(e)$ refers to the *activity* associated to e.
- $\#_{timestamp}(e)$ refers to the *timestamp* of event e.
- $\#_{resource}(e)$ refers to the *resource* of event e.

		Event attributes			Case attributes	
Case id	**Event id**	**Activity**	**Timestamp**	**Resource**	**Vendor**	**Category**
1	11	Purchase request created	2017-04-17 10:11	John	B. Trug	Office supplies
	12	Purchase request released	2017-04-18 14:55	Maria		
	13	Purchase order created	2017-04-18 17:12	Roy		
	14	Goods receipt	2017-04-29 09:06	Ryan		
	15	Invoice receipt	2017-05-16 06:12	Ryan		
	16	Invoice payment	2017-06-16 23:53	Bot		
2	21	Purchase order created	2017-04-19 17:45	Emily	IT & Company	Computer
	22	Invoice receipt	2017-05-13 15:12	Ryan		
	23	Payment block	2017-05-13 15:13	Ryan		
	24	Goods receipt	2017-05-14 06:15	Emily		
	25	Payment block remove	2017-05-14 06:18	Emily		
	26	Invoice payment	2017-05-15 23:52	Bot		
3	31	Purchase order created	2017-04-18 17:12	Bob	B. Trug	Chairs
	32	Goods receipt	2017-04-29 09:06	Emily		
	33	Invoice receipt	2017-05-16 06:12	Bob		
	34	Invoice payment	2017-06-16 23:53	Bob		

Table 2.1: Simplified example event log of a procurement handling process where each line corresponds to an event.

Often activities in processes have a *life-cycle* [Aal16] and are not atomic operations; they often have a duration, i. e., a start and completion time. The life-cycle of an activity describes its behavior. For example, events in an event log may describe the start or completion of activities. Hence, multiple events may refer to the same activity. For this reason, *classifiers* are introduced. A classifier is a function that associates attributes of an event to an activity label, i. e., the name of the activity used in a process model.

Definition 2.4 (Classifier [Aal16])**.** For any event $e \in \mathcal{E}$, we denote $\underline{e}$ as the *name* of the event.

Although different classifiers can be defined, this thesis mainly uses the activity name as the classifier function for events: $\underline{e} = \#_{activity}(e)$. Note that the classifier can also be applied to sequences $\underline{\langle e_1, e_2, ..., e_n \rangle} = \langle \underline{e_1}, \underline{e_2}, ..., \underline{e_n} \rangle$.

An event log consists of *cases* and cases themselves consists of events. Given a single case, the sequence of corresponding events is called *trace*. Each event within a case reflects the execution of an activity. In the following, we formally define cases, traces, and event logs:

Definition 2.5 (Case [Aal16])**.** Let $\mathcal{C}$ be the set of all possible case identifiers. Similar to events, cases can have attributes. For a case $c \in \mathcal{C}$ and an attribute $a \in \mathcal{A}$: function $\#_a(c)$ yields the value of attribute a for case c. Each case contains a mandatory attribute *trace*: $\#_{trace}(c) \in \mathcal{E}^*$, denoted as $\hat{c} = \#_{trace}(c)$.

Definition 2.6 (Trace [Aal16])**.** A *trace* is a finite non-empty sequence of events $\sigma \in \mathcal{E}^*$ such that each event only occurs once within a trace: $1 \leq i < j \leq |\sigma| : \sigma(i) \neq \sigma(j)$.

Definition 2.7 (Event Log [Aal16])**.** An *event log* is a set of cases $L \subseteq \mathcal{C}$ such that each event only occurs at most once in the log, i.e., for any $c_1, c_2 \in L$ such that $c_1 \neq c_2 : \hat{c}_1 \cap \hat{c}_2 = \emptyset$.

Every event and every case is associated with a *unique* identifier. An identifier $e \in \mathcal{E}$ refers to a specific event and an identifier $c \in \mathcal{C}$ refers to a specific case. This is necessary because multiple events may have the same attributes, e. g., events may occur multiple times for different cases. Similarly, multiple cases can occur with the same sequence of activities.

Example 2.1

Table 2.1 shows three cases of an event log of a procurement handling process. The log consists of the cases $L = \{1, 2, 3\}$, events $E = \{11, 12, 13, 14, 15, 16, 21, 22, 24, 25, 26, 31, 32, 33, 34\}$ and the attributes

$A = \{\text{Case id}, \text{Event id}, \text{Activity}, \text{Timestamp}, \text{Resource}, \text{Vendor}, \text{Category}\}$. The activity of the event 13 can be obtained as $\#_{\text{activity}}(13) =$ 'Purchase order created'.

The table also shows the values of the additional attributes. For example, $\#_{\text{category}}(1) =$ 'Office supplies' or $\#_{\text{resource}}(13) =$ 'Roy'. Note, that not all cells are filled with values. Some values may only be defined for specific events or are defined as case attributes.

2.2.2 Simple Event Log Notation

Some process mining techniques consider only the activity names of events within an event log. As a simplified representation, a *simplified event log* is introduced which only stores the activity name and the sequence order; all other attributes are removed.

Definition 2.8 (Simple Event Log [Aal16])**.** A simple event log consists only of traces with activity names. Let Σ be a set of activity names, then a simple event log is a set of sequences over the set of activity names with sequences defined as $\sigma \in \Sigma^*$. A simple event log L is a multi-set of traces over Σ, i. e., $L \in \mathbb{B}(\Sigma^*)$.

The simple event log, transformed from the event log introduced in Table 2.1, is shown in Figure 2.1. It shows the three traces t_1, t_2 and t_3. For instance, $t_1 = \langle$Purchase request created, Purchase request released, Purchase order created, Goods receipt, Invoice receipt, Invoice payment$\rangle$.

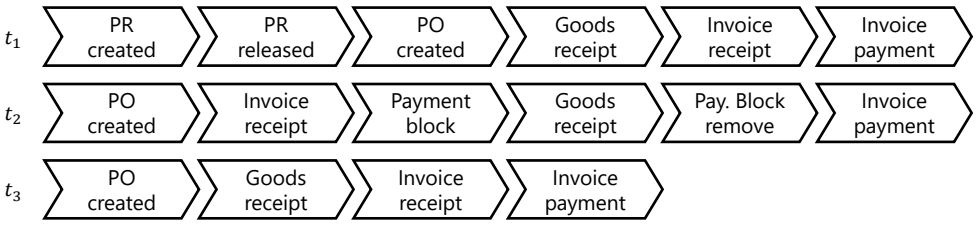


Figure 2.1: A simple event log obtained by transforming the three traces of the example event log shown in Table 2.1.

2.2.3 Process Perspectives

Many different forms of information is needed, besides the activities being executed, to adequately describe an entire business process. Some of this information is also stored in an event log and other information can only be indirectly obtained from the executing organization. In the literature, different

process-oriented modeling perspectives [CKO92; JB96; Aal11; IEE11] as well as goal-oriented perspectives [Pap+16; GA19] are considered:

CONTROL FLOW PERSPECTIVE The control flow perspective deals with the order of activities executed in a process. Most process analysis projects start with the discovery of process models to inspect the actual control flow of a process. Figure 2.4 shows a process model solely showing the control flow of the BPI Challenge 2019 event log.

ORGANIZATIONAL PERSPECTIVE The organizational perspective deals with the organizational structure in which the process is executed. Particularly, it describes the interaction between different resources and their relationship to activities. Resources are not necessarily just people, but also software or hardware robots used for task automation.

TIME PERSPECTIVE The time perspective deals with time-related characteristics of a process. Unlike the control flow perspective, which arranges activities in a sequence, the time perspective extends this view to include information on how long a specific event takes.

FUNCTIONAL PERSPECTIVE The functional perspective deals with the activities or elements of a process. It is a top view of all the elements that are performed, and what informational objects are relevant to a specific case.

DATA PERSPECTIVE The data perspective deals with additional attributes attached to events or cases of a process that further characterize the corresponding element. Data attributes may be of various types and degree of detail. They may also contain routing information about how cases are executed.

GOAL PERSPECTIVE The goal perspective deals with the objectives to achieve from different point of views. From a process mining perspective, these point of views can be divided into the *resource viewpoint*, that refers to the goals behind activities resource (e. g., an employee) are executing, the *process viewpoint*, that refers to the goals of an entire trace, and the *organization viewpoint*, that refers to the overall goal executing the business process [GA19].

The five process-oriented modeling perspectives primarily focus on "how", "what", "where", "who", and "when" related aspects of the process, whereas the goal-oriented perspective focuses on "who", "what", and especially "why" the process is executed [GA19].

2.3 Process Mining

Process mining is a set of techniques which analyze event logs of PAIS to obtain a better understanding of how processes are executed [Aal11]. It

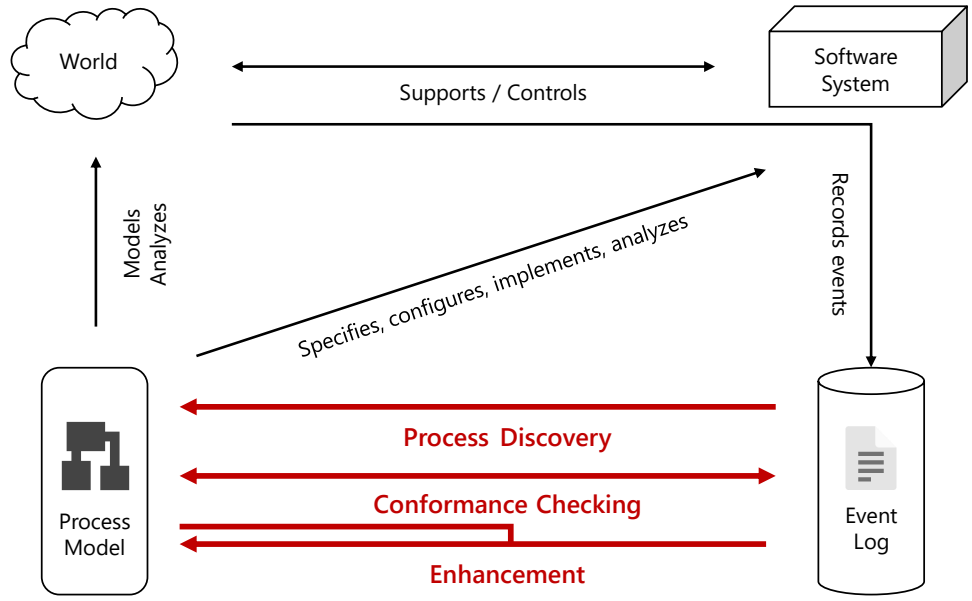


Figure 2.2: Overview of the three tasks of process mining: process discovery, conformance checking, and enhancement. Adapted from [Aal11].

provides valuable insights that help to improve the execution of processes or unveil potential compliance issues. Process mining is typically categorized into three major tasks, as shown in Figure 2.2: process discovery, conformance checking, and enhancement [Aal11].

2.3.1 Process Discovery

Process discovery deals with the automatic construction of an as-is process model solely based on the events observed in an event log. Different algorithms were developed to accurately reconstruct a process model from an event log.

Figure 2.3 shows a cut-out of a the graph automatically constructed from the event log of the BPI Challenge 2019 [Don19]. It is constructed by investigating the directly-follows relationships between activities. Transitions are labeled with the absolute frequency observed in the event log. The graph shows certain patterns, for example, that the activity "Create Purchase Order Item" is directly followed by "Vendor creates Invoice", or that the activity "Record Goods Receipt" is executed several times in a row.

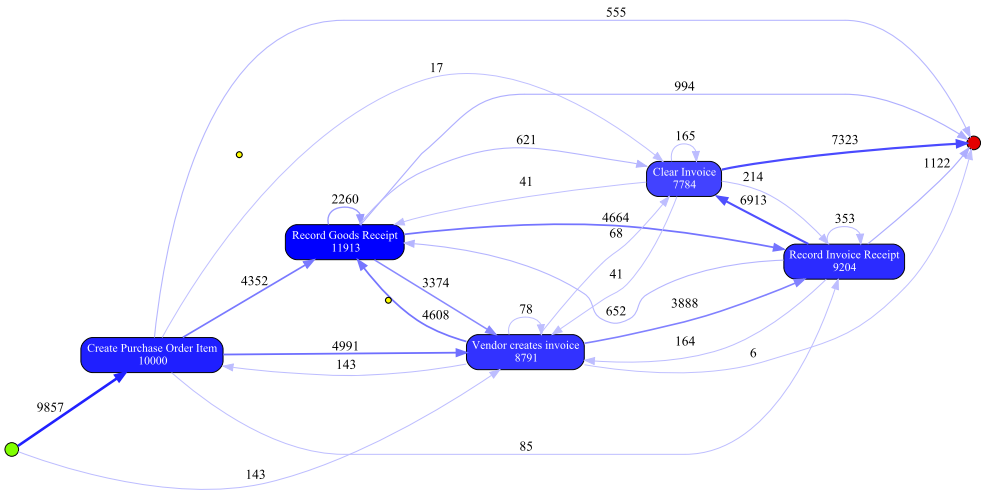


Figure 2.3: Directly-follows graph of a small sample of the BPI Challenge 2019 event log.

However, the directly-follows graph is missing important information, e. g., parallelism of activities. More advanced process discovery algorithms such as the *Inductive Visual Miner* [LFA14a] lead to more informative models. Figure 2.4 shows the process model of the same event log cut-out with the Inductive Visual Miner. The process model indicates parallel activities using the AND-gate, depicted as a diamond with the plus-symbol.

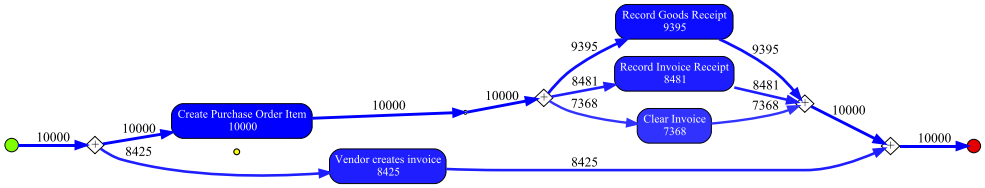


Figure 2.4: Process model obtained with the Inductive Visual Miner [LFA14a] of a small sample of the BPI Challenge 2019 event log.

2.3.2 Conformance Checking

Conformance checking deals with the identification and quantification of deviations between the actual execution behavior in an event log and the modeled behavior in a process model. The process model can be created manually or discovered automatically. Conformance checking aims to relate events of the event log to the activities in the process model with the goal to find commonalities and differences.

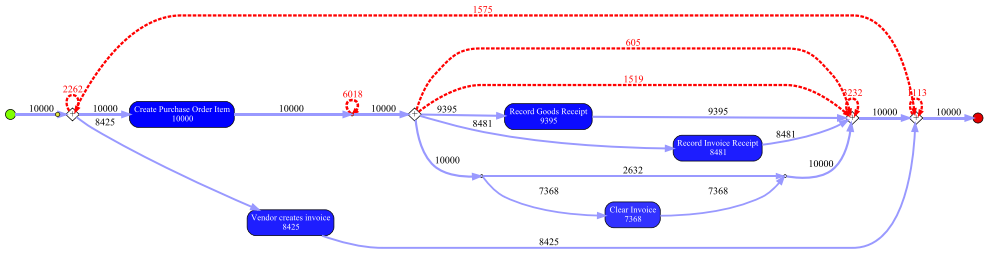


Figure 2.5: Conformance checking result showing deviations on top of a process tree obtained with the Inductive Visual Miner from a small sample of the BPI Challenge 2019 event log.

Figure 2.5 illustrates the deviations between an event log and a process model using the Inductive Visual Miner [LFA14a]. Deviations are indicated by the red dotted lines. For example, we can see that many cases of the event log do not follow the process model and bypass activities of the process model.

2.3.3 Enhancement

In context of process mining, enhancement denotes the task of updating an existing process model based on pertinent information derived from event logs in order to better reflect reality. For instance, the result of conformance checking may be used to update existing process model reflecting the behavior that is observed in reality.

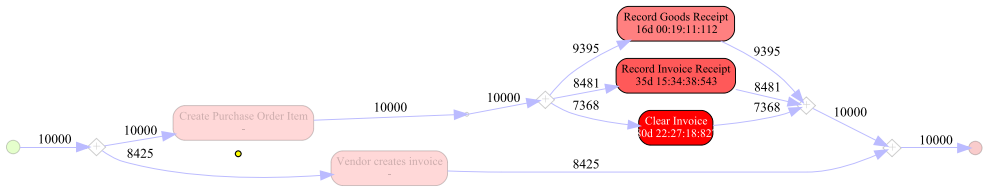


Figure 2.6: Performance overview on top of a process tree obtained with the Inductive Visual Miner from a small sample of the BPI Challenge 2019 event log.

Another enhancement technique is to evaluate the performance of a process by inspecting the throughput times of activities. Figure 2.6 shows the throughput times of the BPI Challenge 2019 event log annotated to the process model. Darker red nodes indicate longer duration times of activities, and lighter red nodes indicate shorter activity duration times.

2.4 Process Discovery

Process discovery automatically constructs process models that describe the actual behavior of a process in an organization. Different from conventional interview-based methods for obtaining such process models, process discov-

ery uses digitally recorded event data from PAISs to reconstruct the actual execution. The underlying assumption of process discovery is that information systems capture events related to a particular case. The goal of process discovery is to construct process models that reflect the actual execution of a process in the most compact and accurate representation.

At the time of writing, process discovery is often the first major undertaking process analysis task because it does not require any prior knowledge about the process under review. It takes an event log as input to discover process models that are visually inspected by analysts. Besides the visual inspection, discovered process models often build the basis for further analysis to obtain interesting knowledge about the executed process.

A wide range of different process discovery algorithms has been developed. A selection of developed process discovery algorithms is listed in the following:

- **α-algorithm**: One of the first workflow pattern based process discovery algorithm was introduced by van der Aalst et al. [AWM04]. The α-algorithm utilizes the directly-follows relation to reconstruct process models.
- **Heuristics Miner**: As the name suggests, the Heuristics Miner [WR11] applies specific heuristics to detect infrequent behavior, parallel executions, loops, and long term dependencies between activities. A detailed description is given in Section 2.4.2.
- **Inductive Visual Miner**: Typically, event logs contain cases that follow many different traces, and some traces are less frequent than others. This infrequent behavior is a challenge for discovery algorithms because including or excluding them affects the quality of the process model (see Section 2.5.1.3). The inductive miner [LFA14a] discovers process models according to the Pareto principle. Essentially, 80% of the observed process behavior can be described by a model that contains only 20% of the model required to describe the entire process behavior. As a result, the process model only contains the most frequent behavior of the event log to describe the process.
- **BPMN Miner**: Unlike other process discovery algorithms that mine flat process models, the BPMN Miner [Con+16] constructs hierarchical process models reflecting the different sub-processes contained in the event log. For discovering the sub-processes, the event log must be extended by an event type attribute indicating the task that produced the event. The sub-processes are then discovered using other flat process discovery algorithms.
- **SplitMiner**: One of the most recent discovery algorithms is called Split-Miner [Aug+17]. It is a fast discovery algorithm that filters the directly-follows graph and detects various combinations of split gateways to

capture concurrency, conflict, and causal relations. Furthermore, it guarantees deadlock-freedom for cyclic process models and soundness for acyclic process models.

A systematic comparison of available process discovery algorithm is given in [Wee+12].

Besides academic process discovery algorithms, a variety of different process mining tools are successfully established in the industry. For example, Fluxicon Disco[1], Celonis[2], and PAFnow Process Mining[3].

2.4.1 Process Map

The simplest form of process discovery is the generation of a directly-follows graph from an event log, also called *process map*. A process map is a directed graph where the nodes represent the activities and the edges represent the transitions between activities. An edge is added to the process map if the corresponding transition is observed at least once in the event log. A process map may contain activity loops if the event log contains traces with a repeating sequence of activities. We define a process map as follows:

Definition 2.9 (Process Map)**.** A process map is a directed graph $P = (N^*, E^*)$ with N^* being the set of activities and E^* the set of transitions between activities. Each process map consists of a unique start activity $s_p \in N^*$ and a unique end activity $e_p \in N^*$. A transition between two activities is a tuple $e_i = (n_{src}, n_{dest})$ with $e_i \in E^*$, $n_{src} \in N^*$, and $n_{dest} \in N^*$.

Every event log can be converted into a process map using the following formalization:

Definition 2.10 (Event Log to Process Map)**.** Let L be an event log according to Definition 2.7, then the process map is defined as $P = (N^*, E^*)$ with

$$
\begin{aligned}
N^* &= \{\underline{(e)} \mid c \in L \wedge e \in \hat{c}\} \\
&\quad \cup \{s_p, e_p\} \\
E^* &= \{(a_1, a_2) \mid c \in L \wedge a_1 = \underline{(\hat{c}(i))} \wedge a_2 = \underline{(\hat{c}(i+1))} \wedge 0 < i < |\hat{c}|\} \\
&\quad \cup \{(s_p, \underline{(\hat{c}(1))}) \mid c \in L\} \\
&\quad \cup \{(\underline{(\hat{c}(|\hat{c}|))}, e_p) \mid c \in L\}
\end{aligned}
$$

1 Commercial tool for process discovery; available at: `https://www.fluxicon.com/`

2 Commercial tool with Business Intelligence (BI) capabilities; available at: `https://www.celonis.com/`

3 Commercial tool for process mining, integrated in Microsoft PowerBI; available at: `https://www.pafnow.com/`

The resulting process map contains all observed activities and transitions of the event log. The start and end activities are explicitly identified by adding two synthetic activities s_p, e_p to the process map.

It is noteworthy that a process map describes far more process behavior than originally observed in the event log which leads to less precise process models (see Section 2.5.1.3). For example, let $L = [\langle a, b, c, d\rangle, \langle a, c, b, d\rangle]$ be a simple event log. This event log produces a process map that also allows the trace $\langle a, c, b, c, d\rangle$ although this trace is not contained in L. Many commercial process mining tools use process maps for visualization because they are typically easier to understand [LFA14b; LPW19] and they can be computed very efficiently, in contrast to other process discovery algorithms.

2.4.2 Flexible Heuristics Miner (FHM)

The Flexible Heuristics Miner (FHM) [WR11] is a robust process discovery algorithm [Wee+12] which inspects the frequency of transitions to generate process models. Essentially, frequent transitions are "kept" while infrequent transitions are "ignored". The discovered process model only contains the most frequent process behavior that is observed in the event log. The resulting process model is called causal net (C-net), it is constructed as follows.

First, the FHM constructs a Dependency Graph (DG) which contains information about the frequency of the directly-follows relationship (see Section 2.4.1).

Definition 2.11 (Direct Successor Relation [WR11])**.** Let Σ be the set of activities, $t \in \Sigma^*$ be a trace of a simple event log $L \subseteq \Sigma^*$, and $a, b \in \Sigma$. Then the direct successor relation $a >_L b$ is defined as follows:

$a >_L b$ if there exists a trace $t = \langle t_1, ..., t_n\rangle$ and $i \in \{1, ..., n-1\}$ such that $t \in L$, $t_i = a$, and $t_{i+1} = b$, i. e., a is directly followed by b.

Then, the DG is constructed by counting the number of directly-follows relationships observed in the event log.

Definition 2.12 (Dependency Measure [WR11])**.** Let L be a simple event log with the set of activities Σ, $a, b \in \Sigma$, and $|a >_L b|$ the number of times $a >_L b$ (direct successor) occurs in L.

$$a \Rightarrow_L b = \begin{cases} \frac{|a>_L b| - |b>_L a|}{|a>_L b| + |b>_L a| + 1} & a \neq b \\ \frac{|a>_L a|}{|a>_L a| + 1} & a = b \end{cases} \tag{2.1}$$

High values of $a \Rightarrow_L b$ indicate a strong dependency relation between activities, whereas low or negative values indicate a weak dependency relation. From the dependency measures, a dependency graph is constructed that

only contains the relations that fulfill a certain threshold, also called the dependency frequency. The default threshold that is widely established is 0.9 [WR11].

Definition 2.13 (Dependency Graph)**.** The causal dependency graph is a tuple $DG = (\Sigma, E)$, where Σ is the set of activities in the simple event log L, and $E \in (\Sigma \times \Sigma)$ the edges that correspond to the directly-follows relation between activities. Each edge in the dependency graph is constructed from the dependency measures described in Definition 2.12 and annotated with the corresponding dependency measure value.

The DG contains information about the dependencies between activities, but it does not describe whether activities are executed in parallel or not. Therefore, a C-net is generated from the DG that characterizes the behavior of a split, i.e., if the split is an AND- or an XOR-split.

Definition 2.14 (C-net [AAD11; WR11])**.** A *C-net* is a tuple (Σ, I, O), where

- Σ is a finite set of activities,
- $I : \Sigma \to \mathcal{P}(\mathcal{P}(\Sigma) \to \mathbb{N})$ is the input frequency function for a given activity,
- $O : \Sigma \to \mathcal{P}(\mathcal{P}(\Sigma \to \mathbb{N}))$ is the output frequency function for a given activity.

A split in the C-net is defined as an input and output frequency function that return the number of times a transition is observed in the event log. For each activity in the C-net that has more than one output activity, the FHM counts if certain output activities are executed together in a trace, or not. Using this information, the FHM decides between an *AND*- or an *XOR*-split:

- If for the majority of traces the output activities are executed within the same trace, the FHM concludes a *parallel*-split.
- Otherwise, the FHM concludes an *XOR*-split.

The same strategy is applied to detect the joins, however, traces are replayed backward. For the sake of brevity other heuristics that further improve the result of the FHM such as the *length-two loops*, *long-term dependency*, or the *relative-to-best* heuristics are not discussed here.

Example 2.2 shows the application of the FHM to a simple event log.

Example 2.2

Let $L = [\langle a, b, c, d, e, g\rangle^{10}, \langle a, c, b, d, f, g\rangle^{10}, \langle a, g\rangle^{3}]$ be a simple event log, containing 23 traces. Table 2.2 shows the number of times one activity is directly followed by another activity, according to Definition 2.11. For instance, a is directly followed by b, $|a >_L b| = 10$.

$\|a >_L b\|$	a	b	c	d	e	f	g
a	0	10	10	0	0	0	3
b	0	0	10	10	0	0	0
c	0	10	0	10	0	0	0
d	0	0	0	0	10	10	0
e	0	0	0	0	0	0	10
f	0	0	0	0	0	0	10
g	0	0	0	0	0	0	0

Table 2.2: Frequency of the directly-follows relation $|a >_L b|$ of the event log L.

The results of applying the dependency measure (Definition 2.12) to the event log L is shown in Table 2.3.

$a \Rightarrow_L b$	a	b	c	d	e	f	g
a	0	0.91	0.91	0	0	0	0.75
b	-0.91	0	0	0.91	0	0	0
c	-0.91	0	0	0.91	0	0	0
d	0	-0.91	-0.91	0	0.91	0.91	0
e	0	0	0	-0.91	0	0	0.91
f	0	0	0	-0.91	0	0	0.91
g	-0.75	0	0	0	-0.91	-0.91	0

Table 2.3: Dependency measure between the activities of the event log L.

The last step is to detect the split and joins in the C-net. Figure 2.7 shows the C-net with the identified AND-splits, indicated by the bindings between transitions. Based on the information stored in the event log L we can observe that activity a is followed by both b and c. In the C-net this is denoted by the frequency output function of activity a, i. e., $O(a) = \{\{b, c\}^{20}\}$. Analogue, we can observe that activity d is followed by either e or f, i. e., $O(d) = \{\{e\}^{10}, \{f\}^{10}\}$.

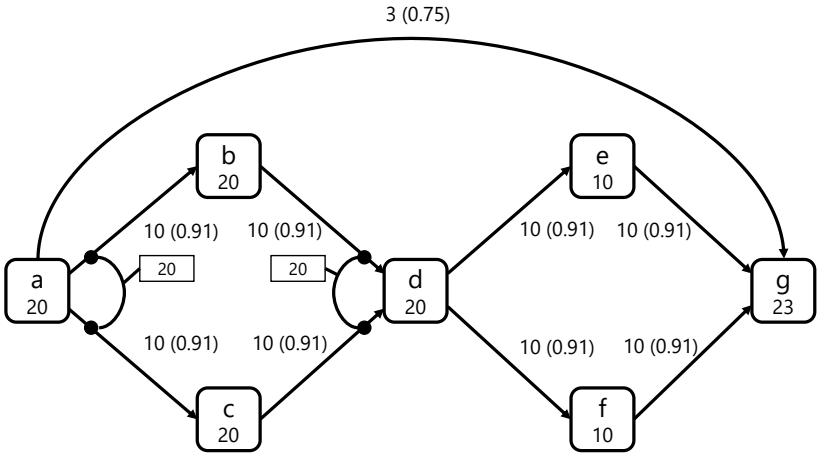


Figure 2.7: C-net derived from the event log L. The nodes indicate the activity and their frequency in the log. The edges refer to the directly-follows relation, labeled by their frequency and dependency measure.

2.4.3 Process Model Quality Dimensions

For evaluating the resulting process model from a process discovery algorithm, four main quality dimensions have been established in the literature: *fitness*, *precision*, *generalization*, and *simplicity* [BDA14; Aal16]. The first three dimensions relate to the quality of the process model with respect to the event log and vice versa. Simplicity measures how easy it is for a human to understand the process model, e. g., how many activities and transitions are contained in the model. Although all four quality dimensions are equally important, they are sometimes conflicting with each other. Typically, all four quality dimensions should be balanced.

The Venn diagram by Buijs et al. [BDA14] in Figure 2.8 compares the behavior described by the process model with the behavior observed in the event log and the actual behavior of the real process, i. e., the system being analyzed. The diagram shows seven areas that can be described as follows: (1) modeled and observed behavior, i. e., system behavior that is observed and described by the model, (2) unmodeled exceptions, i. e., observed behavior that is non-system behavior and not supported by the model, (3) modeled and observed exceptions, (4) modeled but unobserved and non-system behavior, (5) modeled but unobserved system behavior, (6) unmodeled and unobserved system behavior, and (7) unmodeled but observed system behavior.

From this figure, the four quality dimensions are derived which are introduced in the following.

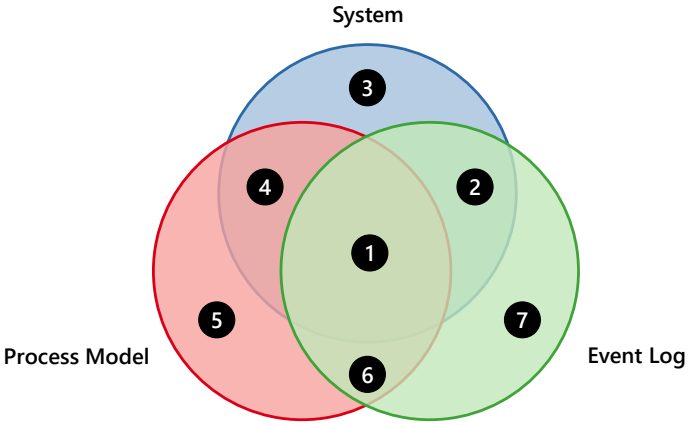


Figure 2.8: Venn diagram illustrating the relations between the behavior of the process model, the observed behavior in the event log, and system's behavior. Adapted from [BDA14].

2.4.3.1 Fitness

The fitness quality dimension measures the ability to replay an event log by the process model, i. e., it is the proportion of observed and modeled behavior $((1)+(2))$ in the total observed behavior $((1)+(2)+(3)+(4))$. If all traces can be entirely replayed by the process model, i. e., if all events and transitions of the event log are described by the process model, this model has a perfect fitness.

Definition 2.15 (Fitness)**.** Let σ be a trace of log L, and M a process model, then $fitness(\sigma, M) \in [0, 1]$. The function $fitness(\sigma, M) = 1$, if the trace $\sigma \in L$ can be entirely replayed by the model (i. e., from the start to the end), and $fitness(\sigma, M) = 0$, if the trace cannot be replayed.

In this thesis, we only consider the control flow of the event log to calculate the fitness of a process model. In order to check if a trace can be replayed by the model, we compute trace alignments [Man+15]. A trace alignment is a mapping between the process model and a trace recorded in an event log. It describes mismatches and matches between the event log and the process model. For example, events may be missing in the process model, denoted as *log moves*, or events may be missing in the event log, denoted as *model moves*. If an event of the event log can be matched with an event in the process model, it is denoted as a *synchronous move*. A trace that can be fully replayed by the process model has a trace alignment containing only synchronous moves.

2.4.3.2 Precision

Precision is a measure indicating how much behavior is allowed by a process model that is not observed in the event log, i. e., it is the proportion of observed and modeled behavior $((1) + (2))$ in the overall modeled behavior $((1) + (2) + (4) + (5))$. While fitness indicates the proportion of traces that can be replayed by the process model, a process model should also precisely describe the observed behavior [MC10]. Specifically, the process model ideally does not allow more behavior than what is observed in the event log. Behavior that is allowed by the model but is not observed in the event log leads to a less precise process model.

This thesis relies on the technique proposed in [MC12]. In order to estimate the precision of a process model, the technique first computes the prefix automaton of an event log. The prefix automaton is a tree that contains information about occurrences of a prefix s in the log L. The second step is to extend the prefix automaton at each step of the traversal with the behavior that is allowed by the model. In particular, transitions that are allowed by the model are added to the prefix automaton. The deviations of the model and the event log are all the transitions in the extended prefix automaton with an occurrence of zero, i. e., they are allowed by the process model but are not observed. These newly added transitions are called escaping edges. Escaping edges that are further branched are called inner escaping edges.

Definition 2.16 (ETC Precision Measure [MC12])**.** Let L be an event log, M is the process model, $\hat{TS}$ is the extended prefix automaton of L with inner and escaping states (I_S^γ, E_S^γ) on a threshold parameter $\gamma \in [0, 1]$. The precision measure is then defined as:

$$precision(\gamma, M) = 1 - \frac{\sum_{s \in I_S^\gamma}(|E_S^\gamma(s)| \cdot s_\#)}{\sum_{s \in I_S^\gamma}(|avail(s, M)| \cdot s_\#)} \tag{2.2}$$

were $avail(s, M)$ is the set of available activities according to the model M after executing the activity sequence s, and $s_\#$ being the number of occurrences of the sequence s.

2.4.3.3 Generalization

Generalization indicates whether a process model is overfitting, meaning that the generated model is very specific to the observed behavior. Event logs are likely to contain only a sample of the process, so a different event log of the same process may produce a different process model [Aal16]. A process model that accurately models the behavior of a process and at the same time replays different event logs of the same process has a high generalization. The calculation of the generalization measure is quite challenging because all possible event logs of the process model need to be considered, depicted as

area (4) of Figure 2.8. In literature, there is no agreement on a measure that adequately measures the degree of generalization. For this reason, this thesis does not investigate the generalization of process models.

2.4.3.4 Simplicity

In contrast to the other three quality dimensions, which measure the accuracy and quality of the process model behavior, simplicity focuses on the structural elements of a process model. In particular, it is always desirable to obtain a process model that is easy to understand but also explains the observed behavior of an event log accurately. However, both objectives are sometimes contradictory, as reality may be complex and difficult to model.

The simplicity of a process model is typically measured by the number of activities and transitions [Aal16]. In graph theory, several measures are used that describe the density of the graph, the in- and out-degree of nodes, and the network degree. These have turned out to be valuable measures for determining the simplicity of the process model and for describing the comprehensibility of process models for humans [MRC07].

In this thesis, the following simplicity metrics are considered:

1. Number of activities $|\Sigma|$
2. Number of transitions $|E|$
3. Inverse Arc Degree [Blu15]:

$$\text{Inverse arc degree} = \frac{1}{1 + \frac{|\Sigma|}{|E|}} \tag{2.3}$$

2.5 Data Mining

In the context of this thesis, data mining is defined as "*the process of discovering interesting patterns and knowledge from large amounts of data*" [HKP11]. Many approaches discussed in this thesis use existing data mining techniques to obtain such interesting patterns and knowledge from event logs. This section introduces cluster analysis (see Section 2.5.1) and frequent pattern mining methods (see Section 2.5.2) which are later used in this thesis.

2.5.1 Cluster Analysis

Cluster analysis or clustering [WFH11; HKP11] is an unsupervised data mining technique which partitions a set of objects into smaller subsets without existing labels. Clustering aims to group objects so that the similarity of objects within each subset is as high as possible, but the similarity of objects from different subsets is as low as possible.

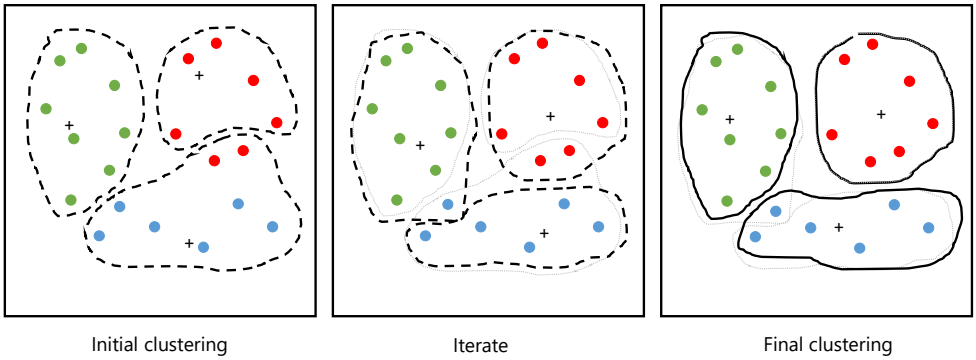


Figure 2.9: Illustration of the k-means algorithm, showing the initial clustering, the iteration step, and the final clustering. The plus (+) indicates the center of a cluster. Adapted from [HKP11].

Identified subsets can be expressed in different ways [WFH11]. Subsets are *exclusive*, i. e., an object is only grouped into a single subset, they are *overlapping*, i. e., an object may be grouped into multiple subsets, they are *probabilistic*, i. e., objects belong to a subset with a certain probability, or they are *hierarchical*, i. e., objects belong to a hierarchically ordered subset. Although a precise categorization of clustering methods is difficult, the major fundamental clustering methods can be classified into four categories [HKP11]: *partitioning* methods (Section 2.5.1.1), *hierarchical* methods (Section 2.5.1.2), *density-based* methods, and *grid-based* methods.

2.5.1.1 Partitioning Methods

Partitioning clustering methods divide a set of objects into a fixed number of exclusive subsets. Let D be a set of n objects, and k be the number of clusters to generate. A partitioning clustering algorithm divides the dataset into k partitions or clusters. Each partition contains objects which are similar to each other, based on a similarity function, but dissimilar to objects in other clusters [HKP11].

The most well-known and widely used partitioning method is *k-means*. It is a centroid-based partitioning method that starts with k centroids and iteratively optimizes the clusters and partitions of the dataset. For a given dataset D of n objects, k-means partitions the objects of D into k clusters $C_1, ..., C_k$ such that $C_i \subset D$ and $C_i \cap C_j = \emptyset$ for $1 \leq i, j \leq k$ (each cluster is a subset of the dataset, and clusters are exclusive). Algorithm 1 shows the main steps of the k-means cluster algorithm.

The k-means algorithm gets the dataset D as well as the number of clusters k as the input. Initially, the algorithm randomly selects k objects from the dataset as a cluster center (Line 1). Next, each object $\vec{d} \in D$ is assigned to the cluster C_i to which its similarity value is the highest (Line 3). To determine

Algorithm 1: k-means Partitioning Algorithm

input : $k \in \mathbb{N}$: the number of clusters
D: the dataset of n objects
output: A set of k clusters

1 k objects from D are selected randomly as the initial cluster centroids.
2 **repeat**
3 assign each object $\vec{d} \in D$ to the cluster $c_i \in C$ with the highest similarity
4 update the cluster center for each cluster
5 **until** *no change*;

the closest cluster, the k-means algorithm calculates the distance $dist(\vec{p}, \vec{c_i})$ for an object $\vec{p} \in C_i$ to the cluster centroid $\vec{c_i}$, where $dist(\vec{x}, \vec{y}) : D \times D \rightarrow \mathbb{R}$ is the distance function (e. g., the Euclidean distance) between two points $\vec{x}, \vec{y} \in D$. After each object $\vec{d} \in D$ is assigned to a cluster, the k-means algorithm updates the cluster centers by building the mean values of the objects assigned to the cluster (Line 4). The process continues as long as the assignment of objects to clusters changes. The algorithm terminates if each object of the dataset is assigned to the same cluster as in the previous iteration. An illustration of the iterative steps is shown in Figure 2.9.

2.5.1.2 Hierarchical Methods

Different from partitioning methods, where each object of a dataset is assigned to exactly one cluster, hierarchical clustering methods [HKP11] organize clusters in different levels of abstraction. These different abstraction levels can be beneficial in scenarios where objects are organized as a hierarchy. Two representative hierarchical methods are *agglomerative* and *divisive hierarchical clustering* (see Figure 2.10) which use a bottom-up or a top-down strategy to assign objects to a cluster. In the following, agglomerative hierarchical clustering is introduced because it is used for the multi-perspective trace clustering algorithm (see Chapter 6).

The fundamental idea of agglomerative hierarchical clustering is that each object is first assigned to an individual cluster. These individual clusters are then iteratively merged based on how similar objects are according to a distance function. Clusters that are closest to each other are merged until all clusters are merged with the root cluster.

There are multiple measures, the linkage measures, to determine the distance between clusters [HKP11; WFH11]:

- **Single-linkage** returns the distance of the objects of the clusters that are the closest together:

$$dist_{min}(C_i, C_j) = \min_{\vec{p} \in C_i, \vec{p}' \in C_j} (|\vec{p} - \vec{p}'|) \tag{2.4}$$

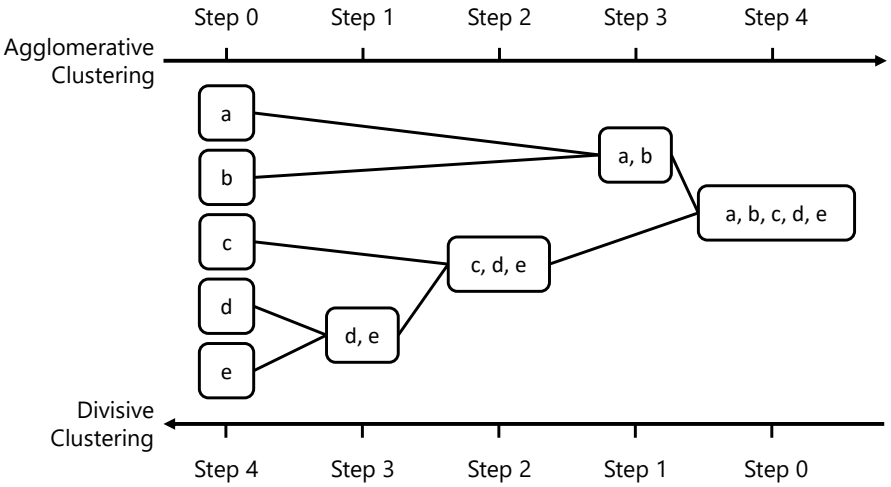


Figure 2.10: Illustration of the two hierarchical clustering methods: agglomerative and divisive hierarchical clustering. Adapted from [HKP11].

- **Complete-linkage** returns the distance of the objects of the clusters that are the most far away:

$$dist_{max}(C_i, C_j) = \max_{\vec{p} \in C_i, \vec{p}' \in C_j} (|\vec{p} - \vec{p}'|) \tag{2.5}$$

- **Centroid-linkage** returns the distance of the mean point of the clusters:

$$dist_{mean}(C_i, C_j) = |\vec{m}_i - \vec{m}'_j| \tag{2.6}$$

- **Average-linkage** returns the average distance between all objects of the one cluster to the objects in the other cluster:

$$dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\vec{p} \in C_i, \vec{p}' \in C_j} |\vec{p} - \vec{p}'| \tag{2.7}$$

- **Ward linkage** calculates how much the sum of squares will increase when clusters are merged:

$$dist_{ward}(C_i, C_j) = \frac{n_i \cdot n_j}{n_i + n_j} |\vec{m}_i - \vec{m}_j|^2 \tag{2.8}$$

$|\vec{p} - \vec{p}'|$ refers to the distance between the two objects $\vec{p}$ and $\vec{p}'$ of the dataset, $\vec{m}_i$ is the mean for cluster C_i, and n_i is the number of objects in a cluster C_i.

2.5.1.3 Clustering Quality Metrics

For evaluating the quality of the resulting clusters, two types of quality metrics can be used, depending on the availability of ground truth labels [HKP11]. In cluster analysis, ground truth is the ideal clustering of the data, often obtained by human experts.

- *Extrinsic methods* (e. g., Adjusted Rand Index, BCubed Precision and Recall) evaluate the result of the algorithm by comparing it with ground truth labels.
- *Intrinsic methods* (e. g., Silhouette Coefficient) evaluate the quality of the clusters by inspecting the separation of clusters if the ground truth is not available.

Adjusted Rand Index The Adjusted Rand Index (ARI) [HA85] is an extrinsic quality metric that compares the objects of the cluster in pairs and solves issues of the original Rand Index (RI) [Ran71; MRS08]. It is computed as follows:

$$ARI(C, C') = \frac{\sum_{ij} \binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}] / \binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}] / \binom{n}{2}} \tag{2.9}$$

where $a_i = |C_i|$ the number of objects in cluster C_i, $b_j = |C'_j|$ the number of objects in the ground truth cluster C'_j and n_{ij} the entry of the contingency table. The contingency table stores the frequency of objects that are in cluster C_i and should be in ground truth cluster C'_j.

BCubed Precision and Recall *BCubed precision* and *recall* are extrinsic quality metrics that fulfill all four essential extrinsic evaluation criteria [HKP11]:

- *Cluster homogeneity*: clusters should only contain objects of the same ground truth cluster
- *Cluster completeness*: objects that belong to the same cluster according to the ground truth should be in the same cluster
- *Rag bag*: objects that cannot be merged with other objects should be put into a separate "miscellaneous" cluster
- *Small cluster preservation*: ground truth clusters with small number of objects should not be further split into smaller clusters than larger clusters

Precision and recall are defined for every object in a clustering. The precision of an object is defined as how many other objects of the same cluster belong to the same class as the object, given by the ground truth.

$$BCubed\ Precision = \frac{\sum_{i=1}^{|D|} \frac{\sum_{\vec{d}_j : i \neq j, C(\vec{d}_i) = C(\vec{d}_j)} correctness(\vec{d}_i, \vec{d}_j)}{||\{\vec{d}_j : i \neq j, C(\vec{d}_i) = C(\vec{d}_j)\}||}}{|D|} \tag{2.10}$$

The recall of an object is defined as how many objects of the same class, given by the ground truth, are assigned to the same cluster.

$$BCubed\ Recall = \frac{\sum_{i=1}^{|D|} \frac{\sum_{\vec{d}_j : i \neq j, L(\vec{d}_i) = L(\vec{d}_j)} correctness(\vec{d}_i, \vec{d}_j)}{||\{\vec{d}_j : i \neq j, L(\vec{d}_i) = L(\vec{d}_j)\}||}}{|D|} \tag{2.11}$$

$L(\vec{d}_i)$ returns the class of $\vec{d}_i$ from the ground truth, and $C(\vec{d}_i)$ returns the cluster identifier of $\vec{d}_i$. Furthermore, the correctness of the relation between $\vec{d}_i$ and $\vec{d}_j$ is defined as:

$$correctness(\vec{d}_i, \vec{d}_j) = \begin{cases} 1 & \text{if } L(\vec{d}_i) = L(\vec{d}_j) \leftrightarrow C(\vec{d}_i) = C(\vec{d}_j) \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

Silhouette Coefficient The silhouette coefficient [Rou87] is an intrinsic quality metric that calculates how similar the objects are to other objects in their subset according to a distance function, compared to the objects of other subsets. It returns a value between -1 and 1, indicating whether the objects fit into their assigned cluster or not. This metric is often used to determine the optimal number of clusters.

It is defined as follows: Let $a(\vec{d})$ be the average distance of an object $\vec{d} \in D$ to all other objects of a cluster A, and let $b(\vec{d})$ be the minimum average distance of $\vec{d}$ to all other clusters. Then, the silhouette is defined as:

$$s(\vec{d}) = \frac{b(\vec{d}) - a(\vec{d})}{\max(a(\vec{d}), b(\vec{d}))} \tag{2.13}$$

The silhouette coefficient is then calculated for each cluster $C_i \in C$:

$$s_{C_i} = \frac{1}{|C_i|} \sum_{\vec{c}_i \in C_i} s(\vec{c}_i) \tag{2.14}$$

2.5.2 Frequent Pattern Mining

Frequent pattern mining is an unsupervised data mining technique that searches for recurring patterns or relationships in a dataset [HKP11]. A frequent pattern is a pattern, such as itemsets (i. e., a set of items), sequences, or structures, that occur frequently in the dataset. For example, in a transaction history of shopping carts, milk and bread would probably be a *frequent itemset* because they are usually bought together.

A set of k items is called an k-itemset, e. g., the set $\{milk, bread\}$ is a 2-itemset. Let $\mathcal{I} = \{I_1, ..., I_k\}$ be a k-itemset and D be a set of database transactions where each transaction T is a non-empty itemset $T \subseteq \mathcal{I}$. We denote all k-itemsets of a transaction database D as L_k. The *support* of an itemset I in D is the percentage of transactions in D that contain all items of I. If the support of an itemset I satisfies a selected *minimum support threshold* (θ), then the itemset is frequent.

Applying frequent itemset mining to large transaction databases is a major challenge because the number of itemsets that satisfy the minimum support threshold can become huge. This is because all subsets of a frequent itemset are also frequent. In order to deal with this issue, the concepts of *closed*

frequent itemsets and *maximal frequent itemsets* are introduced. An itemset X in D is closed if there exists no proper super itemset Y[4] that has the same support as X in D. An itemset X is a maximal frequent itemset of D if there exists no super itemset Y such that $X \subset Y$ and Y is frequent in D.

Example 2.3

Consider $D = \{\{a_1, a_2, ..., a_5\}, \{a_1, a_2, ..., a_{10}\}\}$ as the transaction database containing only two transactions. If we set the minimum support threshold to 1, we find two closed frequent itemsets: $\{a_1, a_2, ..., a_{10}\}$ with the support of 1 and $\{a_1, a_2, ..., a_5\}$ with the support of 2. There is only one maximum frequent itemset $\{a_1, a_2, ..., a_{10}\}$ with the support of 1.

2.5.2.1 Apriori Algorithm

The Apriori algorithm [AIS93] is an iterative algorithm that searches for frequent itemsets in large transaction databases. It uses the k-itemsets to search for (k+1)-itemsets. The algorithm starts to search for 1-itemsets by scanning the database and counting the number of occurrences. If the itemset satisfies the minimum support threshold, the itemset is kept, otherwise, it is removed. After the algorithm found all frequent 1-itemsets, it searches for 2-itemsets by only considering the frequent 1-itemsets. The algorithm terminates if no more k-itemsets are found.

The Apriori algorithm makes use of an important property: "*All nonempty subsets of a frequent itemset must also be frequent.*" [HKP11]. It is based on the observation that the support of an itemset cannot be higher than the support of its super itemset. So if an itemset is not frequent, none of its subsets can be frequent either. This property reduces the search space for frequent itemsets in the transaction database. The Apriori algorithm consists of two steps:

1. **Join**: In the first step, the algorithm generates a set of candidate k-itemsets C_k by joining L_{k-1} with itself. The algorithm only joins itemsets with at least $(k-2)$ equal items, such that only one new item is added to the itemset.
2. **Prune**: In the second step, the candidate itemsets C_k are pruned because they may contain itemsets that are not frequent. All frequent itemsets, i. e., all k-itemsets with minimum support of θ, are included in C_k. To prune the candidate k-itemsets, the Apriori property is used: Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent k-itemset. Thus, any $(k-1)$-itemset of the candidate k-itemset is removed if it is not in L_{k-1}.

4 The itemset Y is a proper super itemset of the itemset X if X is a proper sub-itemset of Y, i. e., $X \subset Y$.

The algorithm terminates if no new k-itemset can be found that satisfies the minimum support threshold. As a result, the Apriori algorithm returns the frequent itemsets with at least a support of θ.

Example 2.4

Let $L_3 = \{\{a,b,c\},\{a,b,d\},\{a,b,e\},\{a,c,d\},\{a,c,e\},\{b,c,d\}\}$ be the set of 3-itemsets and the input of the Apriori algorithm. In the join step, the algorithm joins L_3 with itself. The algorithm only joins pairs of 3-itemsets with at least 2 equal items, which results in the following candidate 4-itemsets set: $C_4 = \{\{a,b,c,d\},\{a,b,c,e\},\{a,b,d,e\},\{a,c,d,e\}\}$. The prune step will remove $\{a,b,c,e\},\{a,b,d,e\},\{a,c,d,e\}$ from C_4 because $\{b,c,e\},\{b,d,e\},\{c,d,e\}$ are not in L_3 and, therefore, cannot be frequent and fulfill the minimum support threshold. The result of the Apriori algorithm is $L_4 = \{\{a,b,c,d\}\}$.

2.5.2.2 FP-Close Mining Algorithm

Different from the Apriori algorithm, which searches for all frequent itemsets in a transaction database, the FPclose algorithm [GZ05] searches for closed frequent itemsets. The method is based on the iterative FP-Growth algorithm [HPY00] that uses a tree data structure, the FP-tree (frequent pattern tree), to maintain the frequency of itemsets. The FP-tree has a root node, labeled with *null*, and for each transaction a new branch is generated as follows: The items of the transaction are sorted by their absolute frequency in the entire transaction database. For each item, a node is generated in T which is linked to its predecessor, i. e., the item within a transaction that is more frequent. The most frequent item in a transaction is linked to the root node of T. Branches which share the same prefix, i. e., the same items in the same order, are merged together by increasing the corresponding frequencies of the prefix nodes. New nodes are generated for each item that is not yet in the tree, and linked to the prefix accordingly.

The FP-close algorithm additionally maintains a header table $T.header$ which stores the frequency of the single items of the transaction database. The header table contains a link to the head of a linked list that points to the corresponding nodes in the FP-tree. Figure 2.11 shows an example FP-close tree with the corresponding header table.

For extracting the frequent itemsets, *conditional FP-trees* T_X are generated from the FP-tree T, containing only itemsets that contain X. The FP-Growth algorithm relies on the principle that the count of the union of two itemsets $X \cup Y$ is the count of the itemsets Y that contain X. Given an item i in the header table, the algorithm follows the linked list that points to the branches in the FP-tree. Following the linked branches to the root node of the tree $(X \cup \{i\})$, reveals the itemsets that contain i. To generate a conditional FP-tree

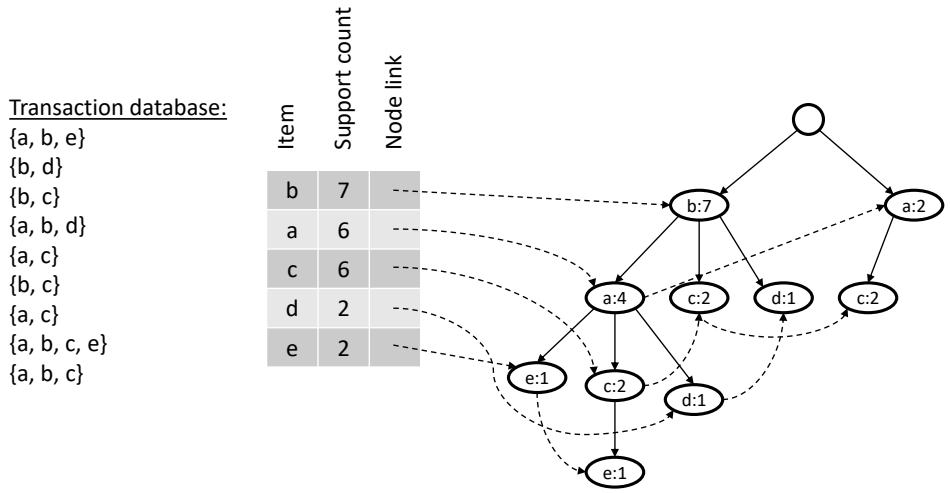


Figure 2.11: FP-close tree for a transaction database with the corresponding header table. Adapted from [HKP11]

$T_{X\cup\{i\}}$ that only contains the frequent itemsets of i, a new header table is generated from all followed branches in T_X that contain i. The header table is only filled with items that fulfill the minimum support threshold. Then, the branches of the FP-tree T_X are visited along the linked list of i to insert nodes and edges in the conditional FP-tree $T_{X\cup\{i\}}$. The FP-Growth algorithm starts by mining the frequent itemsets for the empty itemset, denoted as $T_\emptyset$.

In order to obtain closed frequent itemsets, a separate CFI-tree (closed frequent itemset tree) and a corresponding CFI header table are built for each itemset X, denoted as C_X. Each node in the CFI-tree stores information about the item-name, count, node-link, and level. The level is used for subset testing. The count is used to identify if an itemset is closed or not, i. e., there exists no proper superset which has the same support. Similar to the insertion of itemsets into the FP-tree, itemsets are added to the CFI-tree only if the itemset is closed. Instead of increasing the count of the node, the count is updated by the maximal count. The CFI-tree $C_\emptyset$ and the C_X are merged in each iteration, so the empty CFI-tree contains the closed frequent itemsets after termination.

2.6 Summary

This chapter introduced background knowledge such as basic mathematical notations, expressions, data and process mining algorithms, and the notation for event logs. First, we introduced the notation for multi-sets and sequences which are used to describe event logs. Event logs build the basis for process mining and all contributions of this thesis.

Next, we presented the five process perspectives which are essential to obtain an end-to-end view on a process. In this thesis, we introduce a compliance

checking method (see Chapter 4) that identifies violations with respect to the control flow of a process, based on process maps discovered from event logs. The time perspective is investigated in Chapter 5, which presents a novel process drift detection algorithm to identify points in time when process behavior has changed. Drifts are detected by comparing process models, obtained by the FHM, of different time periods. A multi-perspective trace clustering algorithm, which incorporates the control flow and the data perspective to find cases with similar process behavior, is introduced in Chapter 6. The algorithm analyzes cases attributes by searching for frequent patterns and optimizes the result by investigating discovered process models.

ProcessExplorer integrates the above contributions and guides analysts towards interesting findings, obtained from statistical and machine learning methods. Lastly, Chapter 8 introduces a process optimization algorithm that provides process model adaptation suggestions based on a process map.

Part II

Knowledge Extraction

3

Process Knowledge Artifacts

This chapter introduces the process knowledge artifact framework which builds the foundation for the subsequent contributions of the thesis. The framework consists of several building blocks to support and orchestrate the workflow of analysts in process mining projects. First, the framework closes the gap between available raw data stored in Process-aware Information Systems (PAISs) and the data needed for process mining. Second, our framework enables analysts to capture process analytical knowledge, i. e., the tasks that an analyst conducts to investigate a particular process problem, in a meta-model. In particular, process mining tasks that are often decoupled from each other can be combined to examine processes regarding typical questions. Our framework allows analysts to capture analytic knowledge for reuse, which is not possible with conventional process mining tools.

This chapter is organized as follows. In Section 3.1, a motivating introduction to the process knowledge artifact framework is given. Next, related work in the field is introduced (Section 3.2). Afterward, Section 3.3 presents the results of a requirements analysis, investigating analysis reports of the Business Process Intelligence Challenge (BPIC). Section 3.4 then presents the details of the process knowledge artifact framework. Afterward, Section 3.5 reports the results of a conducted case study. The chapter concludes with the limitations of the framework, and future work (Section 3.6).

3.1 Introduction and Motivation

The analysis of business processes using process mining requires a high level of domain knowledge to obtain valuable insights successfully. Although various efforts [Aal11; Eck+15] have been made to systematize the work of analysts in process mining projects, most analytical tasks are still conducted informally. This is mainly caused by the lack of tool support to systematically capture and maintain process analytical knowledge, e.g., the tasks to evaluate process performance indicators or validate compliance rules. That is why process mining projects are typically supported by external consultants with years of experience in the field [Aal+07].

Let us consider the following scenario, which exemplifies the tasks an analyst may need to perform in a process mining project.

> *Mary is an analyst, interested in analyzing the throughput time between an order and the delivery of goods in a procurement handling process. In particular, Mary wants to know how many cases take longer than 30 days, and why these cases take that long. Before process mining can be applied, she must collect event data of the respective process, which is distributed across different information systems. Mary has to deal with several data sources, transforming data into event logs, and linking them together using manually written scripts to obtain an end-to-end view on the process. In this step, Mary needs to map and convert raw data to activities, events, and attributes. After she has prepared the event log, she can use a process mining tool to load the event log and discover a process model that describes the actual process. Mary needs to understand the activities and the process flow to compute the throughput time of the cases. She can then select the cases that take longer than the desired 30 days. By investigating the attached event and case attributes manually, Mary can identify potential root-causes.*

This simple scenario shows that during the process analysis, Mary has to perform several tasks that cannot be easily performed without a deep understanding of process mining and the process inspected.

Data collection and consolidation. Despite the growth of event data collected in organizations, obtaining suitable event logs for process mining is a non-trivial task. As highlighted in the described scenario, event data is often stored distributed across the organization in relational databases, log files, or other customized formats. Consequently, analysts must extract and prepare event logs manually by specifying the actual events and their relationship to a single process case. Several case studies have shown that data extraction and preparation can consume up to 90% of the entire project time [LL14] because it requires extensive domain knowledge [Aal+07] and it is characterized as a largely manual task.

Process analysis workflow. The actual process analysis workflow depends on the investigated objectives, i. e., interesting aspects of the process. The objectives investigated and the domain knowledge of the analyst typically characterize the analysis tasks performed. Process mining provides a wide range of different analysis methods, which are mostly decoupled, although often being used together. Hence, analysts follow a set of manual tasks or write customized scripts to obtain the desired result. Without a deep understanding of the underlying methods and a systematic workflow, process mining results may be error-prone, unreliable, and hard to reproduce.

In this chapter, we introduce the process knowledge artifact framework, which supports process analysts dealing with the above challenges. The basic idea is to capture and maintain analytical tasks needed to obtain a specific process diagnosis result – from the collection of data to the evaluation of the result –

in a central repository. We introduce a meta-model for process mining tasks that analysts can use to describe how a particular analysis objective can be achieved. Different from manually performed tasks or hand-written scripts, stored analytical tasks in the central repository are executable, reusable, and extensible. Our framework supports analytical tasks, which can be broken down into a specific process problem, but can also improve interactive data exploration.

3.2 Related Work

This section introduces related work in the areas of data collection and consolidation, and process analysis workflow.

3.2.1 Data Collection and Consolidation

Data collection and consolidation is an essential task in process mining to build an event log suitable for analysis. It can take multiple iterations, extensive domain knowledge, and time to obtain and consolidate relevant data from multiple Process-aware Information Systems (PAISs). Extract Transform Load (ETL) [KC11] is a technique commonly used in Business Intelligence (BI) applications to extract, prepare, and consolidate data from different sources for analysis. However, ETL is not specifically designed for process mining which makes transforming relational data structures, that can be found in databases of PAISs, into event logs a major challenge. Adding additional events or attributes requires many manual adjustments to the entire ETL pipeline.

Ontop and OpenSLEX are two approaches specifically designed for process mining. Ontop [Cal+17] is an ontology-based data access system (OBDA) which maps entities of different data sources to domain entities of an ontology to describe events, traces, and attributes of a process. This ontology is specifically designed for process mining to extract event logs from relational data sources. Similarly, OpenSLEX [MRA18] introduces a meta-model to map entities of relational databases to event logs represented in the IEEE standard XES [Ver+11]. It further provides methods to correlate events for building traces and logs. OpenSLEX can build event logs for different process perspectives and recommend case notions. An automated and artifact-centric approach is presented in [NDF13]. The approach automatically extracts event information, case identifiers, and their interrelationships from relational databases by applying schema discovery, summarization, log extraction, and life-cycle discovery.

Obtaining event logs from information systems is still a challenging problem because existing methods either do not provide easy design capabilities to deal with relational data structures or are not capable of abstracting the

technical view. Ontop and OpenSLEX are the two approaches that fulfill both requirements. However, both methods are not interconnected with the related process mining analysis tasks, making it difficult to trace an identified process problem back to the raw data. Furthermore, Ontop requires analysts to deal with ontology and semantic notation.

3.2.2 Process Analysis Workflow

A wide range of academic and commercial process mining tools, such as ProM[1], RapidProM[2], Apromore[3], Fluxicon Disco, Celonis, and PAFnow process mining, exist on the market to support analysts during their work. Most of these tools focus on exploratory analysis, in which results are presented visually. We discuss the details of exploratory analysis in process mining tools in the related work section of Chapter 7.

Some of these tools also allow the automation of simple process mining analysis tasks for the inspection of a specific process problem. In Celonis or PAFnow process mining the calculation of Process Performance Indicators (PPIs) can be automated by designing dashboards with formulas, e. g., investigating high throughput time. It is also possible to check the conformance of event logs according to a process model and automatically identify potential root causes. However, the tasks that can be automated are very limited. The focus of these tools is the exploratory analysis, searching for interesting or suspicious process behavior.

A different approach is the *Business Process Cockpit* [Cas+04], one of the first attempts to automate the analysis of event data. It consists of concepts and architectures for automating ad-hoc analysis tasks, such as evaluating the performance of a process. Instead of visually exploring the data, the system automatically applies data mining techniques to process executions and PPIs to identify potential process problems. Furthermore, the system makes predictions to foresee process problems or areas for optimizations. A meta-model for defining PPIs is PPINOT [Río+13]. The authors propose a framework that automatically evaluates the performance of a process by computing the values of a predefined set of PPIs. Similar approaches are SENTINEL [Ped+08], which uses ontologies, or Process Data Store [SLB03], which offers real-time analysis techniques, to quantify process performance using metrics for process monitoring. However, all these approaches do not provide an end-to-end perspective on the process and neglect the actual

1 Academic open source process mining tool from the Eindhoven University of Technology; available at: `http://promtools.org`

2 Extension to run ProM plugins in RapidMiner; available at: `http://promtools.org/doku.php?id=rapidprom:home`

3 Academic open source process mining tool from the University of Melbourne; available at: `https://apromore.org/`

execution of the process, e. g., analyzing a discovered process model from event logs.

RapidProM [ABZ17] is an extension for RapidMiner[4] to enable process mining capabilities. RapidMiner is a data science platform, enabling the definition of data science pipelines that analyze data with machine learning or data mining algorithms. With RapidProM analysts can automate different process mining tasks by combining different analysis functions. Similarly, analysts can create custom SQL-like scripts in QPR ProcessAnalyzer to execute a limited set of process mining tasks, e. g., data extraction from PAISs, data transformations, and the execution of process discovery. Both approaches automate certain process mining tasks but these tasks cannot be linked to a specific process problem that may be of interest to the analyst. These tools lack a comprehensive methodology to maintain this automation.

Although existing work in this field allows the automation of specific tasks, these automation workflows are not linked to a particular process problem. Different from the related work, our process knowledge artifact framework orchestrates process mining tasks for either visual exploration or for identifying specific process problems that can be defined beforehand the analysis.

3.3 Process Mining Tasks Analysis

We analyzed three years (2017 – 2019) of analysis reports of the Business Process Intelligence Challenge (BPIC) to get a better understanding of what types of analysis tasks are used by analysts to identify potential process problems. The results of the analysis provide the basis for the meta-model of our process knowledge artifact framework (see Section 3.4.2.1). The annual BPIC is part of the Business Process Intelligence Workshop at the Business Process Management conference series. Each year a real-life event log from industry is published with the objective to answer specific questions about the underlying process. Analysts from academia, students, and professionals are invited to analyze the event log and submit their findings. The best submissions are awarded by a jury.

3.3.1 Methodology

We followed the qualitative content analysis guidelines of Mayring [Mayoo] to inspect the BPIC reports. In total, we inspected 41 analysis reports from 154 different authors. Unlike Klinkmüller et al. [KMW19] who investigate visual representations or Lopes et al. [LF19] who investigate general methods

4 Commercial platform for data science and machine learning; Website: `https://www.rapidminer.org/`

and techniques, we are interested in the actual tasks, analysts have executed to answer a specific business question.

We group the tasks that analysts performed into six different categories: *general*, *control flow*, *organizational*, *time*, *functional*, and *data*. Tasks that share multiple process perspectives are categorized into general. The other five categories correspond to the five different process perspectives introduced in Section 2.2.3.

3.3.2 Results

The results for each category are reported in Table 3.1. We list only those tasks that occurred in at least two BPIC reports.

Task	2019	2018	2017	Total
General				
Evaluate logical expressions	12	1	4	17
Perform root-cause analysis	8	2	4	14
Apply clustering algorithms	6	1	7	14
Apply classification algorithms	5	2	6	13
Perform predictive analytics tasks	1	2	7	10
Apply statistical significance testing	2	1	7	10
Perform conformance checking with model	6	1	3	10
Perform conformance checking with rules	7		1	8
Calculate correlations	1		2	3
Rename activities	2		1	3
Detect anomalies in process	2			2
Compare multiple cases	2			2
Control flow perspective				
Discover process model from event log	15	2	23	40
Analyze order of activities	15	3	20	38
Filter cases by control flow	12	1	22	35
Analyze start and end activities	14	2	16	32
Analyze sub-processes of the process	12	3	10	25
Analyze process variants	8	1	15	24
Analyze rework activities	12	1	10	23
Group cases by control flow entities	1		10	11
Calculate fitness for process model	7		3	10
Compute the happy-path of the process	6		4	10
Calculate simplicity for process model	5		2	7
Detect process drifts in control flow	3	2	2	7
Calculate precision for process model	2		2	4
Analyze cases by first-time-right criteria	2	1		3
Compare multiple process models	1		1	2

Task	2019	2018	2017	Total
Organizational perspective				
Analyze activities regarding resources	14	1	21	36
Analyze human vs. automatic activities	11	1	14	26
Analyze cases with external entities			14	14
Group cases by resources/organizational entities	4	1	9	14
Filter cases by resources/organizational entities	7	1	3	11
Perform social network analysis	4		6	10
Analyze organizational entities	6			6
Analyze segregation of duty	3		1	4
Time perspective				
Calculate throughput time of activities/cases	14	2	23	39
Aggregate time-related entities	12	2	21	35
Analyze activity timestamps	12	1	17	30
Group cases by timestamp/throughput time	5	2	12	19
Filter cases by timestamp/throughput time	7	1	7	15
Analyze cases within a time span	2			2
Functional perspective				
Analyze the activity occurrence within cases	15	3	23	41
Count number of cases/events/activities	10	1	16	27
Analyze number of data object occurrence			21	21
Filter cases by functional entities	5	1	14	20
Group cases by functional entities	2	1	5	8
Count the activity co-occurrence	4			4
Data perspective				
Aggregate event/case attributes	15	3	23	41
Filter cases by event/case attributes	15	3	15	33
Group cases by event/case attributes	12	2	14	28

Table 3.1: Tasks performed during process analysis grouped by BPI Challenge.

In the following, we report the details of each category.

3.3.2.1 General Tasks

The *general* category mainly consists of tasks evaluating logical expressions (e. g., simple mathematical calculations), performing root-cause analysis, applying clustering or classification algorithms (e. g., searching for similar cases or finding relations between attributes), performing predictive analytics (e. g., predicting the outcome of a case), applying statistical significance testing, or checking conformance. Conformance checking is performed in two different ways, either a process model is constructed manually, or the analysts defined a set of rules. Less commonly used process mining tasks are

calculating correlations, renaming of activities, detecting anomalous cases, or comparing multiple cases with each other.

3.3.2.2 Control Flow Perspective Tasks

In the category *control flow*, almost all BPIC reports discover a process model using a process discovery algorithm. These process models are usually analyzed regarding the order of activities, the start and end activities, the sub-processes the process may have, the different process variants, and the rework activities (e. g., activities that are executed multiple times). Besides analyzing the characteristics of the process model, analysts filter or group cases by the control flow. For process models metrics such as fitness, precision, or complexity are sometimes calculated to evaluate the quality of the discovered process models. However, these are not used to answer a specific business question. Furthermore, analysts calculate the happy-path (e. g., the process variant that occurs most often in the event log), try to detect process drifts over time, or analyze the first-time-right criteria, i. e., a perfect execution of the process.

3.3.2.3 Organizational Perspective Tasks

The *organizational* perspective deals with the entities participating in the process or executing activities. Here, BPIC reports show that analysts examine the resources involved, analyze the activities of people or automated systems, or analyze the parts of the case that involve external entities (e. g., customers, or vendors). Cases are grouped or filtered by resources and organizational entities. A task that we found only in the years 2019 and 2017 of the BPIC is social network analysis, which analyzes the dependencies between resources in the organization (e. g., handover). Finally, the analysts also analyzed whether the responsibilities for certain activities are divided in a compliant manner.

3.3.2.4 Time Perspective Tasks

The main tasks performed regarding the *time* perspective are the calculation and aggregation of throughput times of activities or cases. Analysts group or filter cases by timestamp, throughput time, or time span.

3.3.2.5 Functional Perspective Tasks

The *functional* perspective mainly deals with the number of occurrences of activities within cases, and the number of cases, events, and activities. Furthermore, in 2017 of the BPIC analysts investigated the occurrence of specific data entities (e. g., the number of orders that are attached to a case). Analysts also filter or group cases by the functional entities. Finally, the number of activity co-occurrence was investigated in the BPIC 2019.

3.3.2.6 Data Perspective Tasks

The last perspective, *data*, mainly deals with the aggregation of event and case attributes, and the filtering or grouping of cases by attributes.

3.4 Framework

In this section, we introduce the process knowledge artifact framework. It supports analysts during their process analysis work with process mining in two aspects: data collection and consolidation, and process analysis workflow. The basic idea of our framework is derived from the observation that many process mining projects start from scratch without reusing previous work. However, the analysis of a specific process often focuses on the same aspects. For example, in a procurement handling process analysts typically check if purchase orders are approved before they are sent to the vendor, or they check if invoices are paid on time to avoid discount lost. Although the process may be executed differently in organizations, a large set of business questions are still the same.

We propose the concept of Process Knowledge Artifacts (PKAs), which are small and self-contained components that store and maintain process analysis knowledge of a particular process problem. The idea is that a PKA stores information about what raw data fields of PAISs are needed, how data needs to be transformed for process mining, what analysis tasks are required to identify a particular process problem, and how results of process mining algorithms are evaluated. Typical process problems can be modeled as a PKA for automated evaluation. A PKA consists of two independent abstraction layers to describe process-relevant knowledge:

1. The **data collection and consolidation layer** stores information about the PAISs involved in the process. Particularly, we introduce a data transformation meta-model that transforms data from multiple heterogeneous data sources into a single event log for the analysis with process mining (see Section 3.4.1).
2. The **process analysis workflow layer** stores information about the tasks that must be executed to inspect the specific process problem of a PKA. We introduce a process analysis meta-model that captures analytic knowledge for a wide range of different process problems (see Section 3.4.2).

Different from SQL scripts or ETL pipelines, PKAs are much more flexible and adjustable because they only investigate a particular process problem. We abstract the raw data transformation from the analysis, which enables the analysis of different PAISs executing the same process without changing the process analysis tasks. Furthermore, PKAs are stored in a central repository,

which is a glossary of process analysis knowledge and maintains process analysis tasks for future applications.

In the following, we describe the two abstraction layers in detail.

3.4.1 Data Collection and Consolidation Layer

The first layer deals with the collection and consolidation of data stored in PAISs. Similar to the OpenSLEX approach, a PKA defines a mapping between the raw data structure of the analyzed information system and the event log structure in process mining.

3.4.1.1 Data Sources and Business Object

We introduce the concept of *business objects* to better reflect what is stored in PAISs. Typically, a process consists of different objects, e. g., purchase orders, invoices, or production items, that are inspected, which we will call business objects. Each business object can produce events, e. g., a purchase order can be created and approved, and have different attributes, e. g., a purchase order has a value. Process mining does not have such a perspective on the process and operates on a flat data structure, consisting of cases and events.

We propose a simplified data source structure, derived from OpenSLEX [MRA18], that consists of tables, fields, and relationships to describe the data transformation meta-model. In the rest of this section, we assume that the data is stored in tabular form, e. g., in a relational database.

Definition 3.1 (Data Source)**.** Let the data source be a tuple $DS = (T, F, tblFld, val, REL)$ such that:

- T is a set of table names,
- F is a set of field names,
- $tblFld \in T \to \mathcal{P}(F)$ is a function that maps a table name to a set of field names,
- $val \in F \to \mathcal{P}(V)$ is a function that maps field names to a set of values,
- $REL \in \mathcal{P}(F) \to \mathcal{P}(F)$ is a function that maps a set of field names to another set of field names, describing the relationships between tables.

A data source can have multiple business objects which store information about the occurring events and attributes, characterizing the object. A business object is defined as follows:

Definition 3.2 (Business Object)**.** A business object is a tuple $BO = (CId, EV, AT, TB)$ such that

- $CId \subseteq F$ is a set of field names that correspond to the case identifier,
- EV is a set of event objects,
- $AT \subseteq F$ is a set of field names that correspond to the case attributes,
- $TB \subseteq REL$ is a set of relations among tables.

Each business object can have multiple case identifiers, multiple definitions of event objects, and case attributes. Events of a corresponding business object are defined in a separate data structure:

Definition 3.3 (Event Object)**.** Let TS be the set of all timestamps. An event object is a tuple $E = (TSFld, timestamp, R, EA, RS, CD)$ such that

- $TSFld \subseteq F$ is a set of field names that refers to the timestamp of the event,
- $timestamp \in TSFld \rightarrow TS$ is a function that maps a set of timestamp fields to a timestamp,
- $R \in F$ is a set of field names that refer to the resources of the event,
- $EA \subseteq F$ is a set of field names that refer to additional event attributes of the event,
- $RS \subseteq REL$ is a set of relations among tables,
- $CD \in \mathcal{P}(F) \rightarrow \{true, false\}$ is a condition function that returns true if the values of the given fields fulfill the condition, otherwise false.

The above-introduced meta-model allows analysts to define events and attributes for specific business objects. For generating event logs, the framework automatically evaluates the business objects by generating corresponding SQL statements that transform the data from PAISs to flat event logs. Two data tables are generated for each business object, the case attribute table, and the events table. Example 3.1 illustrates the definition of business objects as well as the final results for a procurement handling process.

> **Example 3.1**
>
> We consider a procurement handling process in SAP. Purchase orders are stored in the tables *EKKO* (purchase order head), *EKPO* (purchase order positions), and *EKBE* (purchase order history). *EKKO* stores the main information of a purchase order, *EKPO* stores the individual positions of the purchase order, and *EKBE* stores any changes to the purchase order such as goods receipt. The tables are linked by the purchase order number *EBELN* and the position number *EBELP*.

EBELN	ERNAM
..12926	MUSTERMANN
..12966	MUSTERMANN

Table 3.2: Excerpt of the EKKO table of SAP MM.

EBELN	EBELP	AEDAT	TXZ01	NETWR	WERKS
..12926	00010	2019-04-29 13:39	Monitor	149,99	IT
..12926	00020	2019-04-29 13:39	Keyboard	39,95	IT
..12966	00010	2019-04-29 13:40	Printer paper	4,99	OFFICE

Table 3.3: Excerpt of the EKPO table of SAP MM.

EBELN	EBELP	CPUDT	CPUTM	ERNAM	VGABE	SHKZG
..12926	00010	2019-04-30	15:43	MUSTER..	1	S

Table 3.4: Excerpt of the EKBE table of SAP MM.

In this example, the purchase order is considered as the object of interest. We specify the business object *Purchase Order* and consider two events *Purchase order created* and *Goods receipt* which are extracted from the three tables (see Table 3.2 - Table 3.4).

The following events are specified:

1. $E_{PO\_Created} = (TSFld, timestamp, R, EA, RS, CD)$ with
   - $TSFld = \{EKPO_{AEDAT}\}$
   - $timestamp = id$
   - $R = \{EKPO_{ERNAM}\}$
   - $EA = \{\}$
   - $RS = \{\}$
   - $CD = true$

   and *id* being the identity function.
2. $E_{Goods\_receipt} = (TSFld, timestamp, R, EA, RS, CD)$ with
   - $TSFld = \{EKBE_{CPUDT}, EKBE_{CPUTM}\}$
   - $timestamp = dat\_format$

- $R = \{EKBE_{ERNAM}\}$
- $EA = \{\}$
- $RS = \{EKPO \leftrightarrow EKBE\}$
- $CD = \begin{cases} true, & \text{if } val(EKBE_{VGABE}) = 1 \\ & \wedge\ val(EKBE_{SHKZG}) = \text{'S'} \\ false & \text{otherwise} \end{cases}$

and dat_format being the function that concatenates date and time to a timestamp.

The business object *Purchase Order* is then defined as $BO_{PurchaseOrder} = (CId, EV, AT, TB)$ with

- $CId = \{EKPO_{EBELN}, EKPO_{EBELP}\}$
- $EV = \{E_{PO\_Created}, E_{Goods\_receipt}\}$
- $AT = \{EKPO_{TXZ01}, EKPO_{NETWR}, EKPO_{WERKS}\}$
- $TB = \{EKPO \leftrightarrow EKKO\}$

For the case identifier, the purchase order number and the position of the item are concatenated. Both events are attached to the business object, and additional case attributes (purchase item text, price of the purchased goods, and the plant) are added. Lastly, the business object also stores the relationships between the tables which are defined in the original data source.

As a result, the following two flat tables are generated:

CaseId		Event	Timestamp	Resource
..12926	00010	PO_Created	2019-04-29 13:39	MUSTERMANN
..12926	00020	PO_Created	2019-04-29 13:39	MUSTERMANN
..12966	00010	PO_Created	2019-04-29 13:40	MUSTERMANN
..12926	00010	Goods_receipt	2019-04-30 15:43	MUSTERMANN

Table 3.5: Resulting event log table of the Purchase Order business object.

CaseId		TXZ01	NETWR	WERKS
..12926	00010	Monitor	149.99	IT
..12926	00020	Keyboard	39.95	IT
..12966	00010	Printer paper	4.99	OFFICE

Table 3.6: Resulting case attribute table of the Purchase Order business object.

3.4.1.2 Issues with Multiple Objects

We mentioned earlier that process mining does not consider multiple objects within a case, which leads to two issues:

1. If multiple objects refer to a single case, the resulting log contains a repetition of events because it is not possible to distinguish between the events. This repetition leads to self-loops in the process model (e. g., if multiple invoices correspond to a single purchase order, all invoice events occur multiple times).
2. If multiple cases refer to a single object, all events of this object are duplicated for each case (e. g., if multiple orders are paid with one payment, the payment transaction is multiplied by the number of orders).

Our framework tries to overcome these issues by extracting events for each business object individually before combining multiple objects. Events of each business object are stored in a separate data table. Business objects are linked together via a *lookup table*, storing the unique identifiers of business objects that belong together within a case. In essence, the case lookup table contains a row for each link and a column for each business object.

> **Example 3.2**
>
> Let us assume we investigate a process with two business objects: a purchase order and an invoice. If a single purchase order is linked to two invoices, the lookup table contains two entries. The first entry consists of the identifier of the purchase order and the identifier of the first invoice. The second entry consists of the identifier of the purchase order and the identifier of the second invoice.

Depending on the perspective needed, the analyst decides how cases are built together. For example 3.2 two different perspectives can be chosen. First, purchase orders and invoices should be merged, which means only a single case is generated, leading to a loop in the reconstructed process model because the invoice event is triggered twice. Second, purchase orders and invoices should be kept separate, that means two cases are generated, leading to a duplication of the purchase order. Figure 3.1 shows the dependencies between the business object and the result tables.

3.4.2 Process Analysis Workflow Layer

The second layer deals with the actual analysis of the event log. We distinguish between two types of analysis:

- **Targeted analysis** focuses on the identification of well-known process problems using predefined criteria. The main objective is to check a set

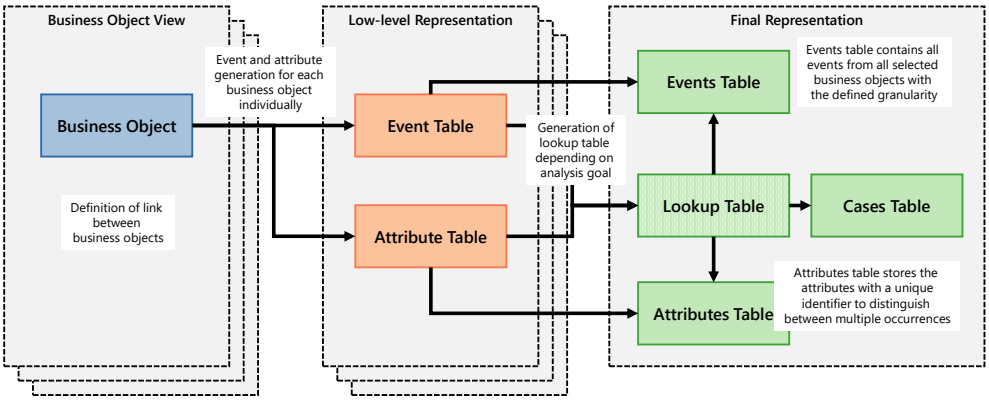


Figure 3.1: Schematic structure of the generation of an event log from raw data sources using business objects.

of process problems that occur typically in this specific type of process (see Section 3.4.2.1).

- **Exploratory analysis** focuses on the detection of unknown behavior and characteristics of the process that the analyst is not aware of. The main objective is to find interesting patterns in an unsupervised or exploratory fashion (see Section 3.4.2.2).

Both types of analysis tasks are supported by the process knowledge artifact framework. The following subsections describe the details of both analysis types.

3.4.2.1 Targeted Analysis

The goal of the targeted analysis is to identify well-known and frequently occurring process problems. This type of analysis often refers to the manual analysis work of consultants using process mining tools. Because such process problems have been observed in many organizations, they are well understood and can be identified by following a predefined set of tasks. In other words, the analyst already knows what he needs to analyze [Cas+04].

From the report analysis of the BPIC (see Section 3.3) we designed a meta-model for commonly used process mining analysis tasks based on popularity. This meta-model is used in a PKA to define the tasks required to investigate a particular process problem. Similar to the manual workflow of analysts, tasks in a PKA are organized as configurable pipelines with an arbitrary number of tasks that are executed one after another. Each task produces an output which is stored in a result container, allowing the use in subsequent tasks to construct complex analysis tasks.

We divide the process analysis workflow tasks into two classes:

Perspective	Analysis Task	Function
All	Filtering	Filter
	Aggregation	Aggregation
	Group by	GroupBy
Organizational	Resource cardinality	OC.Resource
	Binding of duty	OC.BindDuty
	Segregation of duty	OC.SegregationOfDuty
	Resource attribute rules	OC.Expression
Control flow	Process model	CC.Discovery
	Order of activities	CC.DirectlyFollows
		CC.EventuallyFollows
	Start and end activities	CC.BeginsWith
		CC.EndsWith
	Data-driven activity precondition	CC.Expression
Time	Activity duration	TC.EventDuration
	Case duration	TC.ProcessDuration
Functional	Activity coexistence	FC.Coexist
	Activity occurrence	FC.Occurrence
	Data-driven activity existence	FC.DataExists
Data	Attribute evaluation	DC.Expression

Table 3.7: Targeted analysis tasks that can be used in a PKA.

FILTERING, AGGREGATION, AND GROUP-BY TASKS The first class of tasks refers to the filtering of cases, aggregation of entities, and grouping cases according to an expression. According to our report analysis, we found that these tasks were performed across the five process perspectives.

BUSINESS RULE TASKS The second class of tasks refers to the validation of certain business rules, e. g., activities are executed in the desired order, or the duration time between activities is within a certain range. The business rule tasks performed in the reports are also widely discussed in related work [San+96; CVB13; GHV08; MBA13; DMA12; Man+15], which we use to systematize the tasks being supported by our framework.

Table 3.7 shows the analysis tasks supported by the process knowledge artifact framework.

Example 3.3 illustrates the definition of a PKA.

Example 3.3

This example demonstrates the definition of a PKA which examines long duration times between the activities *PO_created* and *Goods_receipt* of a procurement handling process. In particular, we are interested in cases for which the duration between the two activities takes longer than 30 days. Both events correspond to the *Purchase order* business object which is defined in the previous example 3.1. The PKA stores a link to the corresponding business object such that the required data source is known to the PKA.

The actual computation of the duration time between the two activities is performed by two workflows. The first workflow discovers a process model by applying a process discovery algorithm. The second workflow of the PKA computes the actual duration time between the events and compares them with the threshold of $\geq$ 30 days. One could also investigate the duration time of cases that differ from the mean value, i. e., finding outlier cases. Affected cases are returned in the *affected* result container variable which is used for further inspection and root cause analysis.

Table 3.8 shows the entire definition of the described PKA.

Property	Component / Configuration	Result
References	*BO-PurchaseOrder*	
Pipeline[0]	CC.Discovery	*pm.model*
	perspective: 'Control-flow'	
Pipeline[1]	TC.EventDuration	*dur.order*
	input: *pm.model*	
	start event: 'PO_created'	
	end event: 'Goods_receipt'	
	Filter	*affected*
	expression: '*dur.order* $\geq$ 30 days'	
Affected Cases	*affected*	

Table 3.8: Simplified example PKA investigating in cases with duration times between *PO_created* and *Goods_receipt* longer than 30 days.

3.4.2.2 Exploratory Analysis

The second type of analysis in process mining is the exploratory analysis, which deals with the problem that it is unrealistic to define all relevant process problems. The goal of the exploratory analysis is to find potential process problems that are not covered by the targeted analysis.

In this thesis, we introduce two exploratory analysis algorithms which are discussed in separate chapters:

- In Chapter 5, a process drift detection algorithm is introduced. It deals with the problem that the execution of a process may change over time. Often new regulations, guidelines, or organizational changes indirectly influence the process execution without being recognized or documented. The algorithm investigates changes to the discovered process models to identify such behavioral changes.
- Chapter 6 introduces a multi-perspective trace clustering algorithm that combines control flow and data perspective to find cases with similar process behavior. Different from process drift detection, which analyzes process behavior over time, trace clustering aims to find these different behaviors independent from time.

Besides these two algorithms, we introduce ProcessExplorer (see Chapter 7), an intelligent and interactive visual recommendation system to enable fast data analysis and exploration of large and complex event logs. ProcessExplorer extends existing process mining tools by introducing a recommendation engine that guides analysts towards automatically obtained findings in event logs. Particularly, it suggests interesting subsets of cases and computes insightful insights that analysts are typically interested in.

3.5 Case Study

This section presents the results of a case study in which the process knowledge artifact framework was used to analyze business processes of different organizations. The main evaluation aspects of the case study are threefold:

1. We investigate the reduction of manual work of analysts using PKAs to partially automate their work (see Section 3.5.2).
2. We investigate how PKAs can be reused across different process mining projects, i. e., analyzing PAISs of different organizations (see Section 3.5.3).
3. We investigate how our framework compares to the state of the art process mining tools (see Section 3.5.4).

The case study was performed in cooperation with a consulting company that uses process mining to analyze business processes in organizations. The process knowledge artifact framework was used in five different process mining projects (see Table 3.9) to investigate the procurement handling process of SAP systems.

Dataset	Industry	Timespan	Variants	Instances
A	Public Transportation	27 Month	~37 320	~568 000
B	Public Media	12 Month	~8 600	~29 000
C	Business Consulting	15 Month	~850	~12 000
D	Technology	36 Month	~11 800	~225 000
E	Production	36 Month	~4 750	~82 000

Table 3.9: Overview of the process mining projects which were used to evaluate the process knowledge artifact framework.

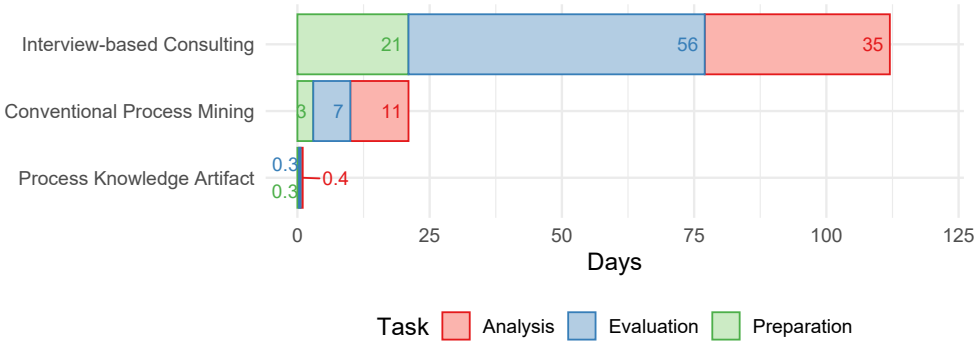


Figure 3.2: Overview of time spent in process mining projects comparing the process knowledge artifact framework and conventional process mining tools. Interview-based consulting results were obtained from former experience of the consulting company.

3.5.1 Experiment Setup

The consultants received a short introduction into the framework, explaining the rough idea of process knowledge artifacts and showing them how PKAs are created and maintained. Together with the consulting team, five different process mining projects were selected in which the framework was used, parallel to the regular analysis with conventional process mining tools. We choose a within-subject design study due to the small number of consultants participating in our study. We measured the time analysts spend to analyze the data with time-sheets for each project. The time required was divided into *data preparation*, *analysis*, and *evaluation* of the results. The result is depicted in Figure 3.2.

3.5.2 Reduction in Manual Work

In this section, we present the results of a user study, evaluating the reduction of manual work of analysts during the analysis of event logs.

In our study, the average total project time with conventional process mining tools was between 3 and 4 weeks. Out of this period, an average of 3 days was spent to transform the raw data into an event log, including the selection of the required data sources, the collection and extraction of the data, and the preparation of an event log. Due to the different scope of the projects, the time required to answer the business questions about the process varied. On average 11 days were spent using a process mining tool to manually investigate the process, find process issues, and answer the desired questions. According to the analysts, most of the time was spent understanding the process, the case attributes, and the terminology. The evaluation and preparation of the results for the customer took an average of 7 additional days to complete the project.

The same projects were performed increasingly faster with each project using the process knowledge artifact framework. During the first projects, most of the time was spent getting familiar with the framework and modeling the required analysis knowledge (see Section 3.5.3). However, the subsequent projects benefit from the knowledge modeled in PKAs. Instead of performing the same set of tasks again, analysts reused knowledge stored as PKAs and simply executed them to obtain the desired result. For example, analysts build a set of PKAs dealing with compliance issues, e. g., segregation of duty or service level agreements. Nevertheless, analysts could modify and adjust PKAs, i. e., events, case attributes, and pipelines, to reflect the differences between projects. As a result, the last of the five projects was completed in 2 days, including data preparation, analysis, and result evaluation.

Although the comparison showed a significant improvement of the framework over conventional tools, the differences in the projects make it difficult to report exact numbers. The results of the case study revealed that modeling process analytic knowledge into a meta-model, i. e., our PKA structure, can make analysis workflows more efficient. Typical process problems can be evaluated with less manual work. In this study, we did neither investigate the quality of the analysis results nor did we compare the results of our approach with conventional process mining tools. This would have been useful to see if the manual analysis is comparable to the results derived with our framework.

It should be noted that the focus of this study was the targeted analysis with PKAs. We also conducted a separate user study that investigated exploratory process analysis and our recommendation engine to guide analysts towards interesting findings automatically (see Chapter 7).

3.5.3 Knowledge Modeling

This section investigates the knowledge modeling using PKAs. The case study revealed that defining specific process analysis tasks is an unfamiliar task for analysts, rather than visually inspecting the event log. Although the

actual questions and analysis tasks were known, analysts had difficulties defining analysis tasks in the form of workflows without immediately seeing the results visually. Consequently, analysis tasks were defined in an iterative manner, which led to a trail-and-error approach until the desired outcome was achieved. Analysts spend a lot of time searching for the correct data attributes, activities, and operators because input support was limited. We also found that analysts had to switch back and forth between the editor and the results page of the analysis because they repeatedly executed the analysis to validate their input. As a result, the analysis found the methodology confusing because visual feedback of workflows was not given during the design phase. Unlike conventional process mining tools, where modeling of analysis tasks is basically performed in an exploratory fashion during the analysis itself, our framework forced analysts to model their tasks in advance.

Besides the difficulties analysts had, we also observed that the analysts became more and more familiar with our framework and created an increasing number of PKAs after they experienced the reduction of manual tasks. In particular, analysts spent about 10 days modeling and designing PKAs for the procurement handling process across all projects. In total, 104 PKAs were designed with 48 evaluating simple data expressions (e. g., aggregating prices, calculating costs), 38 calculating PPIs (e. g., total discount lost, service level agreements), 10 investigating the order of activities (e. g., checking simple business rules such as maverick buying), and 8 calculating the throughput time between activities.

The results of the case study revealed that analysts designed a wide range of different PKAs investigating different process problems. Analysts positively commented that almost all their required tasks were supported by our framework and that the generated PKAs could be easily reused in subsequent projects. However, visual exploration of event logs is still an essential task in process mining. The lack of instant result presentation was a major concern of all analysts which makes it unnecessarily complicated and time-consuming to model analytic knowledge.

3.5.4 Comparison with the State of the Art

In this section, we compare the capabilities of the process knowledge artifact framework with conventional process mining tools. It should be noted that the comparison is not exhaustive and only gives an rough overview of the capabilities. We compare our framework with ProM, RapidProM, Fluxicon Disco, QPR ProcessAnalyzer, and Celonis Intelligent Business Cloud (IBC). Table 3.10 shows an overview of the tools and capabilities compared.

	Process Knowledge Artifact	ProM	RapidProM	Fluxicon Disco	QPR ProcessAnalyzer	Celonis IBC
(1) Data Collection & Consolidation	●	◐	●	○	●	●
(2) Process Discovery	●	●	●	●	●	●
(3) Process Analysis Workflow	●	○	●	○	●	◐
(4) Result Evaluation	●	○	○	○	○	◐
(5) Knowledge Repository	●	○	◐	○	○	◐

Table 3.10: Overview of the different process mining tools compared with the process knowledge artifact framework. ● indicates the presence, ◐ indicates a partial presence, and ○ indicates the non-presence of a capability.

3.5.4.1 Data Collection & Consolidation

Before applying process mining techniques, data collection and consolidation is needed to obtain a suitable event log. These tasks are typically not integrated into conventional process mining tools although being similarly important. Only RapidProM, QPR ProcessAnalyzer, and Celonis allow transforming raw data into event logs. RapidProM allows analysts to define workflows to extract, transform, and validate data sources and prepares them for process mining. An SQL-based approach is provided by QPR ProcessAnalyzer which allows the design of custom scripts and the import of preprocessed event logs. Celonis offers a graphical interface to define the event collection from databases, flat files, and other PAISs. All other tools rely on the existence of an already prepared event log, either stored as a CSV or XES file. Our framework integrates data collection and consolidation as a core component because identifying different process problems may need different data sources. For better orchestration of the data sources, our approach links raw data sources to the process problems, i. e., each PKA stores which data it needs for inspection.

3.5.4.2 Process Discovery

Process discovery is supported by all process mining tools, either producing a process map (e. g., ProM, RapidProM, QPR ProcessAnalyzer, Celonis, and our framework) or generate process models with AND and OR-gate semantics (e. g., ProM, RapidProM, and Fluxicon Disco).

3.5.4.3 Process Analysis Workflow

Since almost all conventional process mining tools focus on interactive visual exploration, most of the tools do not provide many capabilities for automating analysis workflows. RapidProM is the only tool that also allows designing workflow pipelines to automate certain process analysis tasks. Different from our framework, the workflow pipelines designed in RapidProM are not easily adaptable. Modifying interconnected tasks immediately leads to an inconsistent and infeasible pipeline. QPR ProcessAnalyzer allows analysts to write SQL scripts with process mining capabilities and custom extensions. Similar to RapidProM, maintaining ad-hoc SQL queries is error-prone and complicated. Celonis recently introduced operational applications that in essence contain a workflow to analyze certain tasks automatically. However, there is not much information about which tasks can be automated and how flexible these applications are. Our framework provides process mining specific tasks that can be easily combined within a workflow. Each PKA consists only of tasks specifically required to investigate a particular process problem, facilitating reuse and customization.

3.5.4.4 Knowledge Repository

The key idea of our framework is the knowledge repository in the form of PKAs. They store knowledge about typical process problems and how they can be investigated. In most process mining tools, knowledge about the analysis cannot be easily consolidated and reused in subsequent process mining projects.

Celonis and RapidProM are the only two investigated conventional process mining tools that allow to store and reuse process analysis knowledge. In Celonis analysts can design a dashboard, consisting of interactive charts and PPIs that are automatically updated for different event logs. However, dashboards do not necessarily provide immediate evaluation results to process problems. RapidProM allows storing analysis workflow pipelines which can be executed on different event logs. Still, maintaining and modifying such analysis workflow pipelines is complicated because tasks strongly depend on each other, such that modifications may break the entire pipeline.

3.6 Discussion and Limitations

The process knowledge artifact framework provides several building blocks for targeted and exploratory analysis. However, there are some limitations and research directions for future work.

3.6.1 Meta-Model

The proposed meta-model for data extraction assumes the existence of an exhaustive data model. In particular, the relationships among data fields and tables must be known. In certain situations this information is not available, leading to manual exploration and identification of the underlying data structure. Our method does not yet provide automatic detection of such relationships.

3.6.2 Targeted Analysis

The proposed PKA allows analysts to design analysis pipelines to automate the computation of targeted analysis tasks. This concept assumes a well-understood process for which specific tasks can be easily defined. However, in some process mining projects, the overall analysis goal is unknown and it is the goal of the project to find interesting patterns. Targeted analysis cannot help in these situations and analysts must rely on exploratory analysis.

The conducted case study revealed that defining targeted analysis workflows without direct visual feedback is complicated in certain situations. During the design phase of PKAs, it is necessary to provide some visual feedback for the analyst to validate the correctness of their designed workflow.

3.6.3 Case Study

An extended case study could deliver more in-depth insights into the proposed methodology and framework. Certain design aspects of the process knowledge framework, such as the user interface or the runtime performance, were not considered in the case study. In particular, useful capabilities of other process mining tools, i. e., dashboards, operational applications, exploratory analysis, were also not considered. It might also be interesting to investigate the analysis workflows that are followed by analysts to obtain more insights about how process issues are analyzed. Further research is also needed to compare the outcome of predefined with exploratory analysis, which was not investigated in our case study.

3.7 Conclusion

In this chapter, we introduced the process knowledge artifact framework which proposed meta-models to conduct and automate workflows for analysis tasks in process mining. In summary, the main contributions of this chapter are:

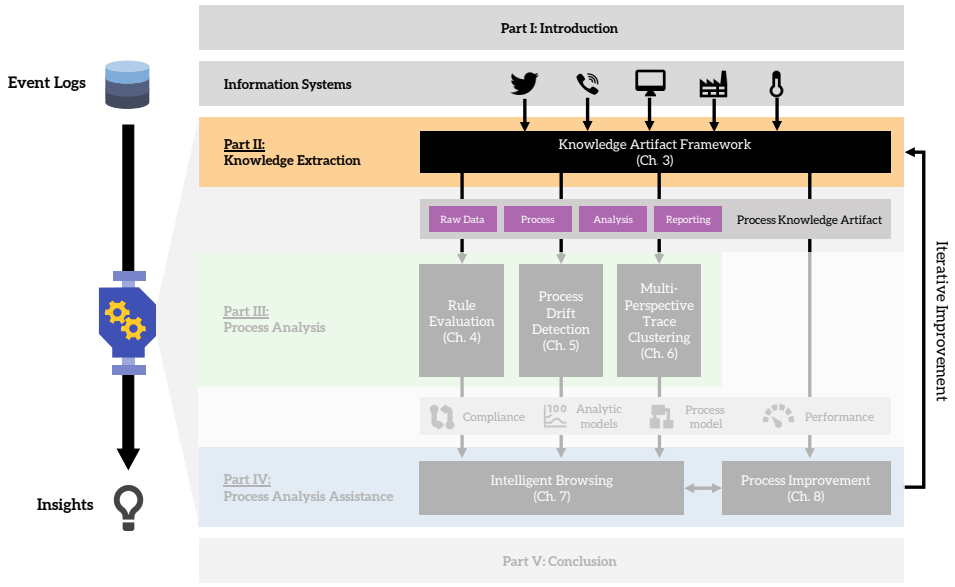


Figure 3.3: Overview of the chapter and contributions.

1. A meta-model for automatic collection and consolidation of event logs from heterogeneous data sources in a generic reusable way.
2. A process analysis framework that allows the capturing of process analytic knowledge in PKAs. Typical process problems can be modeled and easily investigated for different event logs.
3. Results from a case study of five different process mining projects that strongly support the claimed advancements made by the process knowledge artifact framework.

The process knowledge artifact framework builds the basis (see Figure 3.3) for collecting and consolidating heterogeneous data from PAISs. The presented meta-model bridges the gap between data stored in PAISs and event logs required for process mining. The framework orchestrates the tasks being executed by analysts for targeted analysis to investigate typical process problems and supports exploratory analysis. In the following chapters, algorithms and methods are presented to obtain insights regarding process compliance and process behavior from event logs.

Part III

Process Analysis

4 Business Rule Evaluation using TFA

The previous chapter introduced the foundational framework for process analysis and divided analysis tasks into targeted and exploratory ones. Targeted analysis tasks evaluate either specific business rules or Process Performance Indicators (PPIs). This chapter introduces a targeted analysis algorithm for identifying process compliance violations in large event logs. In particular, we adapt concepts of taint flow analysis to validate specific control flow constraints in business processes. Taint flow analysis originates from the software engineering research community, where it is used to find vulnerabilities in software programs by constructing a graph reachability problem. It was shown that this problem can be computed very efficiently [RHS95].

This chapter is organized as follows. First, Section 4.1 gives a short introduction into compliance checking for processes. Next, related work in the context of backward compliance checking is discussed (Section 4.2). In Section 4.3 a compliance checking algorithm based on graph-reachability is introduced. Then, Section 4.4 compares the contributed algorithm with other related work in an experimental evaluation. The chapter concludes with the limitations of the presented work, and potential future work (Section 4.5).

> **Publication:** This chapter is based on the following publication:
>
> Alexander Seeliger, Timo Nolle, Benedikt Schmidt, and Max Mühlhäuser. "Process Compliance Checking using Taint Flow Analysis." In: *Proceedings of the 37th International Conference on Information Systems - ICIS '16*. 2016.
>
> **Contribution Statement:** I led the idea generation, implemented the prototype and performed the data evaluation. *Timo Nolle, Benedikt Schmidt* and *Max Mühlhäuser* supported the conceptual design and contributed to the writing process.

4.1 Introduction and Motivation

Process compliance checking is an essential task for organizations to reveal risks that could have a negative impact on the organization's regulatory compliance and profitability [CVB13]. Consequently, organizations are in-

terested identifying potential process issues quickly to comply with regulations, policies, and guidelines. On the one hand, several external authorities force organizations to be compliant with certain business rules such as the General Data Protection Regulation (GDPR) [Tan16] or the Sarbanes-Oxley Act [SO02]. Violations against such regulations may cause legal proceedings and unexpected costs. On the other hand, organization's shareholders and other stakeholders demand frequent and independent assessments to prevent non-compliant activities. Process compliance checking helps organizations to identify violations and to implement measures to reduce risks early.

Compliance checking is commonly divided into two categories [RFA12; FZ14]:

- **Forward compliance checking** prevents compliance violations by verifying process models at design time and implementing mechanisms to monitor compliance at runtime. Typically, the financial sector uses forward compliance checking to enforce regulations strictly [Bec+14]. In other industrial sectors that might not be necessary and would decrease the flexibility of processes, reducing the ability to quickly respond to external influences, such as seasonal demand changes.
- **Backward compliance checking** investigates recorded event logs from Process-aware Information Systems (PAISs) to detect and locate non-compliant process behavior. Unlike forward compliance checking which is applied at design and runtime, backward compliance checking is applied after the process has been executed, e. g., in the context of audits [Aal+10; WGN13].

In the following, we focus on backward compliance checking.

Process mining allows analysts to obtain a more in-depth view on the compliance of processes in PAISs by analyzing recorded information about the actual process execution. A major challenge is to compare the prescribed process behavior, i. e., a process model or a set of compliance rules, with the observed behavior in an event log to detect and locate non-compliant behavior. For example, a wide range of conformance checking approaches has been proposed to check conformance given a process model (e. g., Petri nets, Workflow nets, or BPMN) and an event log [CW99; GCC09; MC10; Knu+10; Wei+11; DMA12; AAD12; IEE11; MCA13; Sch15; Man+15]. These approaches replay the event log on the process model [Aal11; AAD12] or compute trace alignments [RFA12; Man+15] to detect and locate conformance issues. Another approach is to define a set of compliance rules, e. g., in the form of Linear Temporal Logic (LTL) expressions [ABD05], and compare them to an event log.

Existing backward compliance checking approaches have several issues. First, describing compliant process behavior as a process model or formulating expressions in temporal logic can be difficult and is not well supported in existing approaches [RFA12]. Often up-to-date and usable process models

are not available [FZ14; Has+18] and dealing with complex temporal logic constructs can be difficult for end users. Second, the high computational complexity of compliance checking approaches, even with a small set of rules, might make the application unfeasible [Has+18; DMA12]. However, the growing availability of event data from PAISs and the increasing complexity of processes require new approaches to overcome these challenges. Lastly, most compliance checking approaches do not provide useful insights into the detected violations, such as provided by forward compliance checking methods.

This chapter introduces a backward compliance checking algorithm that transforms the problem of business process compliance checking into a graph-reachability problem to address the above issues. The work in this chapter is based on the taint flow analysis used in software engineering, e. g., to detect security vulnerabilities in the program code of interactive websites [Gua+11], Android apps [Arz+13], etc. The fundamental idea of taint flow analysis is that any program variable in software that can be modified by an outside attacker (e. g., a variable set by an input field on a website) leads to a potential security vulnerability. If that value of the variable is used in an expression to set another variable, this other variable may also pose a risk. Variables that contain potential risks and are not sanitized before a sensitive command is executed are marked as a potential security vulnerability. The taint flow analysis constructs a directed graph of program variables to identify variables with a potential security risk.

Similar to the flow of variable values in program code, traces in an event log can be converted into a process map, containing nodes and edges like a regular directed graph (see Section 2.4.1). This process map is the input for the taint flow analysis algorithm, which searches for realizable paths that violate a set of predefined rules. The basic idea is to introduce *conformance-establishing activities* that mark a realizable path in the process map as compliant, i. e., these activities act like sanitizers. A realizable path consists of sensitive producer activities, e. g., a purchase order, and a sensitive consumer activity, e. g., a payment of an invoice. Whenever such a realizable path exists in the process map that does not contain a conformance-establishing activity, this path is not compliant. Reps et al. [RHS95] have shown that the computation of such realizable paths in graphs can be solved efficiently using the tabulation algorithm. Our approach focuses on two control flow compliance checks: the existence of activities and the order of activities. Additionally, the algorithm provides detailed diagnostics about the violation, not usually given by related approaches. It highlights the violating path in the process map and returns a counterexample that violated the prescribed business rule.

4.2 Related Work

This section introduces related work in the context of backward compliance checking. There are two fundamentally different approaches to backward compliance checking: (1) approaches using process models, and (2) approaches using rules as a description language.

We compare the related work based on a set of relevant requirements for designing backward compliance checking algorithms:

(R1) **Use of a comprehensive modeling language**
The modeling language for specifying the compliance checks should be easy to use and comprehensive for analysts [RFA12].

(R2) **Provision of visual explanations**
The algorithm should provide visual explanations that describe a detected violation [DMA12; RFA12]. This allows the analyst to visually perceive the violation in the corresponding process model.

(R3) **Support for partial process descriptions**
Often, in real-life scenarios, the full description of a process is not available, i. e., the behavior of all process participants is not known or cannot be controlled [Mag+11]. Therefore, the algorithm should not assume that analysts can provide the entire process description.

(R4) **Support for large event logs**
Real-life event logs tend to become very huge with a large number of cases. The compliance checking algorithm should be able to handle large real-life event logs [Don+17].

(R5) **Provision of proven correctness**
The validity of the underlying algorithm for compliance checking should be proven to identify compliance violations correctly.

Table 4.1 compares the relevant research in each group regarding the established requirements. The following sub-sections elaborate on the related work in detail.

4.2.1 Process Model-based Compliance Checking

An extensive research area in the field of process mining is conformance checking which compares predefined *process models*, e. g., Petri nets, Event-driven Process Chains (EPCs), Workflow nets, or BPMN, with event logs obtained from PAISs [CW99; FZ14; Has+18]. These process models often need to be manually crafted from textual documentation or are obtained from reference models [GCC09]. Due to the different granularity of the activities in the process model and in the event log (e.g., different activity names or technical events) an activity matching is needed. In this case, algorithms that

	Use of a comprehensive modeling language	Provision of visual explanations	Support for partial process descriptions	Support for large event logs	Provision of proven correctness
Process Model-based Compliance Checking					
Cook and Wolf [CW99]	●	○	○	●	○
Gerke et al. [GCC09]	●	○	◐	●	●
Muñoz-Gama and Carmona [MC10]	●	◐	○	●	◐
Knuplesch et al. [Knu+10]	●	●	○	○	●
Weidlich et al. [Wei+11]	●	●	●	●	◐
de Leoni et al. [DMA12]	◐	●	●	○	●
van der Aalst et al. [AAD12]	●	○	○	○	◐
van der Aalst et al. [IEE11]	●	●	○	○	●
Muñoz-Gama et al. [MCA13]	●	○	○	●	●
Schönig [Sch15]	●	○	●	●	◐
Mannhardt et al. [Man+15]	●	●	○	○	●
van Dongen et al. [Don+17]	●	●	○	◐	◐
Rule-based Compliance Checking					
van der Aalst et al. [ABD05]	○	○	●	○	●
Award et al. [ADW08]	●	○	●	○	●
Award et al. [AW09]	●	●	●	○	●
Governatori and Rotolo [GR08]	●	○	○	○	●
Ly et al. [LRD10]	●	○	●	○	●
Maggi et al. [Mag+11]	●	●	●	◐	●
Ramezani et al. [RFA12]	●	●	●	○	●

Table 4.1: Property comparison of process compliance checking algorithms. ● indicates the fulfillment, ◐ indicates a partial fulfillment and ○ indicates the non-fulfillment of a requirement.

match the activity names of the model with the event log [BMW14; Bai+17] or abstract low-level events [Man+16] are applied.

A common method to compare event logs with process models is to replay the recorded traces on the prescribed process model [Aal11; AAD12; Man+15]. The degree of deviation is quantified with the fitness measure [AAD12], indicating how many of the observed traces can be successfully replayed by the process model. Other metrics such as precision are also used to determine the deviation between a given event log and a process model [MC10]. In [Wei+11] different compliance metrics based on behavioral characteristics are introduced. Other methods check conformance on different perspectives [Knu+10; Man+15].

Trace alignment, inspired by biological sequence alignment, is another approach for obtaining deviations between a predefined process model and an event log [JA12; Man+15; Don+17]. Although trace alignment has become the state of the art for conformance checking in process mining [RFA12], it is not trivial to compute alignments for large event logs [Don+17]. In [MCA13], the authors propose a decomposition approach that divides a large process model into its sub-processes to speed up the computation. Another approach [Don+17] deals with this issue by balancing between alignment quality, i. e., an optimal solution, and efficiency.

For processes that are not particularly structured, declarative process models are used for compliance checking. DECLARE [PSA07] is a declarative process modeling language that allows specifying compliance rules. Event logs are verified against DECLARE models by computing trace alignments [DMA12]. In [Sch15], a database-based method using SQL expressions is introduced. The authors propose specific SQL queries for detecting compliance violations such as control flow constraints, resource assignment constraints, or cross-perspective constraints.

Process model-based compliance checking can be easily applied in situations where a prescribed process model is available because no new process modeling language is needed. However, due to the increased flexibility of processes, process models are often not maintained anymore because it is not feasible to update models continuously. In addition to the necessity of a comprehensive or partial process model, it is computationally complex to obtain deviations between a process model and an event log.

4.2.2 Rule-based Compliance Checking

A different research direction is rule-based backward compliance checking [CVB13]. Instead of comparing process models with event logs, a set of business compliance rules is specified and validated. In organizations,

the definition of business rules is typically more approachable than the specification of the exact process behavior in form of a model.

LTL expressions allow specifying a wide range of different and complex compliance rules [ABD05]. The specification of business rules in LTL is quite difficult for non-experts. For this reason, a simplified formal definition of LTL was introduced in [ADW08]. This approach was further simplified by introducing a visual query language, called BPMN-Q [AW09]. Analysts can specify rules that check the presence, absence, and order of activities. These rules are then automatically transformed into regular LTL expressions. A compliance framework based on LTL and colored automata is introduced by Maggi et al. [Mag+11].

In addition to LTL expressions, other domain-specific languages for compliance checking are proposed. In [GR08], a formal contract language is introduced that allows expressing normative specifications of a process. With SeaFlows [LRD10], process independent compliance rules can be defined in a graph representation. These independent rules are then verified by mapping them to the actual process. A collection of 55 Petri net compliance rule patterns is proposed in [RFA12]. The approach maps the compliance rule patterns to the inspected process and uses trace alignment for temporal pattern verification.

In contrast to process model-based methods, in which comprehensive or partial process models must be created to describe the prescribed process behavior, rule-based methods can explicitly check for prohibited behavior or validate given requirements. Creating and maintaining business rules is usually easier. Similar to process model-based approaches, validating business rules is a highly complex task that may take a long time to compute for large event logs.

4.3 Compliance Checking as a Reachability Problem

In this section, we introduce a backward compliance checking algorithm that transforms the problem into a graph-reachability problem. The algorithm searches for non-compliant paths in the process map, instead of comparing the event log with the desired process model or using a model checker for evaluating LTL expressions. The algorithm consists of three parts:

1. An *exploded supergraph* is constructed to validate the set of compliance rules (Section 4.3.2). The exploded supergraph maintains all non-compliant paths that correspond to each compliance rule defined.
2. A *tabulation algorithm* (Section 4.3.3) computes the reachability of nodes in the exploded supergraph.

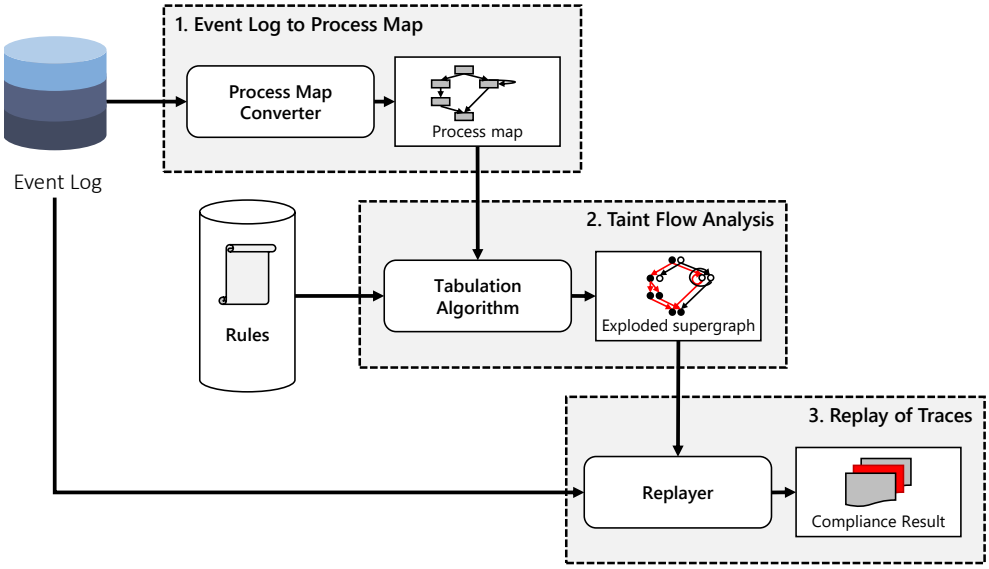


Figure 4.1: Overview of the steps performed during the compliance checking using taint flow analysis.

3. The traces of the event log are replayed on the generated super graph to determine the cases that violate the defined rules (Section 4.3.4).

Figure 4.1 shows the overall architecture and the performed steps of the compliance checking algorithm.

In the following, each step is described in detail.

4.3.1 Taint Flow Analysis

The basic idea of our algorithm is derived from the static analysis of programs, used for finding security vulnerabilities in software applications, also called taint flow analysis. It detects possible security vulnerabilities through analyzing data flows, e. g., JavaScript variables in web applications [Gua+11]. Each variable that can be modified by external sources (e.g., user inputs) must be sanitized before it can be securely used in a data consumer for further processing. Example 4.1 illustrates this case.

Example 4.1

Consider a simple JavaScript web application that shows a text input field. The entered text is printed on the website if the user clicks on a button.

```
var input = document.getElementById('inputtext').value;
document.writeln(input);
```

> Normally, the web application works as expected and shows the entered text. However, if the user enters a modified text that contains executable code, this code is executed as soon as the user clicks the button. User input with malicious code that is not sanitized leads to unexpected behavior.

Our backward compliance checking approach is inspired by the same idea. Instead of modeling compliant process behavior as process models or business rules, we suggest the use of conformance-establishing activities, similar to sanitizers in taint flow analysis. The easiest way to explain the concept is to introduce a simple example (see Example 4.2).

> **Example 4.2**
>
> In a procurement handling process, the order of goods corresponds to a sensitive producer of a potentially non-compliant execution. Each purchase order must be followed by the receipt of goods at a certain point in time. In this case, the delivery of the goods is the conformance-establishing activity in the process. Organizations do not want to pay an incoming invoice for goods that they have never received. Every good should be delivered before the invoice payment, which is the consumer activity in this process.

Generally speaking, processes consist of sensitive consumer activities, e. g., the payment of an invoice, that are relevant for compliance checking. These consumer activities are typically associated with a producer activity, e. g., the creation of a purchase order, that triggers the consumer activity to be executed. For the process to be compliant, a conformance-establishing activity, e. g., the delivery of the purchased goods, must be executed between the producer and the consumer activity. This conformance-establishing activity refers to the sanitizer in taint flow analysis because it marks a process execution as compliant.

The basis of our compliance checking approach is a process map, generated from an event log according to Definition 2.4.1. The process map consists of all activities and transitions that are observed in the event log. Each activity in the event log refers to a node in the process map, and transitions between activities correspond to the edges in the process map. From a process map, a realizable sequence of transitions can be constructed:

Definition 4.1 (Path)**.** Let $P = (N^*, E^*)$ be a process map, then a path from activity m to activity n is a sequence of transactions $\langle e_1, e_2, ..., e_{j-1} \rangle$ for which there is a sequence of activities $\langle n_1, n_2, ..., n_j \rangle$ such that $e_i = (n_i, n_{i+1})$ for $i = 1, 2, ..., j-1$. Note that the target activity of each transition is the source activity of the subsequent transition.

In contrast to a trace, which consists of a sequence of the activities, a path consists of a sequence of transitions. Paths can be compliant, i. e., they either do not contain a producer activity or they contain a conformance-establishing activity or non-compliant, i. e., they contain a producer and a consumer activity without a sanitizer activity in between.

Formally, compliance checking using taint flow analysis is described as a 3-tuple of producer, conformance-establishing, and consumer activities.

Definition 4.2 (Compliance Check)**.** Let CC be the 3-tuple of producer, sanitizer, and consumer activities for a specific compliance check:

$$CC = (P, S, C) \tag{4.1}$$

where P, S, C are three different sets of activities:

1. P is the set of **producer** activities which denote the creation of a non-compliant path in the process map. Each generated non-compliant path must be sanitized by a specific set of sanitizer activities before a consumer activity can be executed conform.
2. S is the set of **sanitizer** activities, i. e., conformance-establishing activities, which mark non-compliant paths as compliant.
3. C is the set of **consumer** activities that are sensitive to non-compliant paths. The corresponding compliance check is violated if there exists a non-compliant path that contains a consumer activity.

Each compliance check defines a path in the process map, where the activity order is specified as $producer \xrightarrow{requires} sanitizer \xrightarrow{before} consumer$. In between the producer, sanitizer, and consumer activities, an arbitrary number of other, even unknown activities, can be executed. The taint flow algorithm evaluates if there exists a realizable path between a producer activity and a consumer activity without having a conformance-establishing activity executed in between. The existence of such a realizable path in the process map indicates a possible violation of the corresponding compliance check. However, due to the construction of the process map, we do not know if the detected violation has been executed. In Section 4.3.4, this specific issue will be addressed. Multiple compliance checks are evaluated by maintaining a path for each check individually. A path is only generated or sanitized by the activities of the corresponding compliance check. This case is illustrated in Example 4.3.

Example 4.3

Consider the procurement handling process where the *Purchase Order* activity generates a non-compliant path. The *Goods Received* activity is the sanitizer, and the *Invoice Payment* is the consumer activity. Figure 4.2

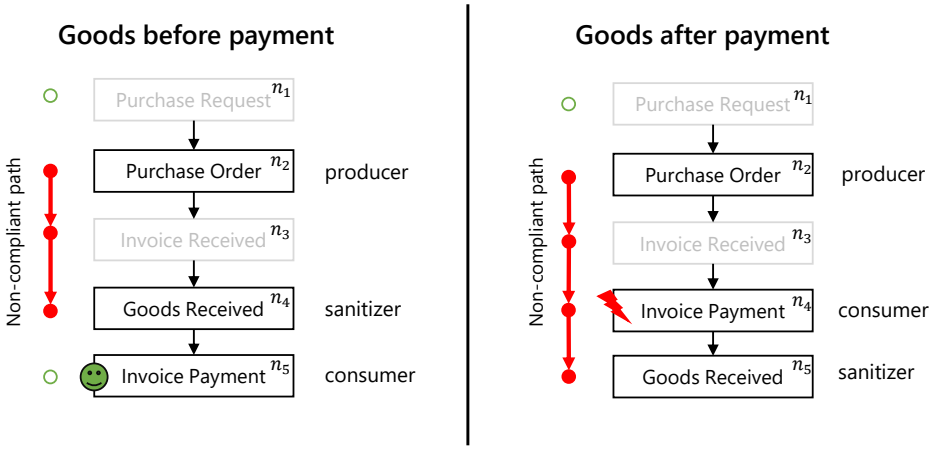


Figure 4.2: Path example for a given process map of a simple procurement process [See+16a].

shows two process maps demonstrating how non-compliant path are maintained. A non-compliant path is depicted as a red path with closed circles left along with the process map.

In both process maps the node n_2, which corresponds to the producer activity, generates a non-compliant path. The node n_3 is neither a producer, sanitizer, nor a consumer activity. In this case, the non-compliant path is forwarded to the subsequent activity. Up to this point, both process maps are equal, but the last two activities are swapped. On the left process map, the non-compliant path is sanitized at node n_4 due to the sanitizer activity *Goods Received*, i. e., it is not forwarded. This sanitizer activity leads to a compliant process execution because there is no realizable non-compliant path between *Invoice Payment* and *Purchase Order*. The right process map shows that the *Invoice Payment* activity is part of a non-compliant path because it is not sanitized beforehand. As a result, a compliance violation is detected.

Reps et al. [RHS95] proposed a framework for transforming different data flow problems into a graph-reachability problem. Instead of searching for realizable paths between producer and consumer activities in the process map, the framework maintains a so-called *exploded supergraph* that contains information about the reachability of nodes. In our approach, the exploded supergraph is the process map "multiplied" by the number of compliance checks, i. e., for each compliance check nodes of the process map are duplicated. Edges in the exploded supergraph correspond to transitions of the process map but are maintained by a distributive function (see Section 4.3.2). The distributive function describes which edges are generated in the exploded supergraph for each transition in the process map and depending on the compliance checks to be evaluated. Due to this construction, the realization

of non-compliant paths in the process map is equivalent to the reachability of consumer activity nodes in the exploded supergraph.

Next, we provide a full definition of our approach based on our extended taint flow algorithm. We highlight the parts where we have extended the original algorithm [RHS95]. The taint flow algorithm in the context of compliance checking is defined as follows:

Definition 4.3 (Taint Flow Algorithm)**.** An instance of a taint flow analysis problem TFA is a four-tuple of $TFA = (P, D, F, M)$, where

1. $P = (N^*, E^*)$ is the process map as defined in Definition 2.9.
2. D is the set of all considered compliance checks as introduced in Definition 4.2.
3. $F \subseteq 2^D \rightarrow 2^D$ is a set of distributive functions that describe the edges in the exploded supergraph between activities.
4. $M : E^* \rightarrow F$ is a function that maps P's transitions to distributive functions.

Distributive functions can be concatenated, i. e., they can be applied to a sequence of transitions, to calculate the corresponding path in the exploded supergraph. The concatenation of a distributive function $f : 2^D \rightarrow 2^D$ is called a path function which is defined over a path q as follows:

Definition 4.4 (Path function)**.** Let $TFA = (P, D, F, M)$ be an instance of a taint flow analysis problem, and let $q = \langle e_1, e_2, ..., e_j \rangle$ be a non-empty path in P. For a given distributive function f, the corresponding path function is defined as $pf_q =_{df} f_j \circ ... \circ f_2 \circ f_1$ for all i, $1 \leq i \leq j$, and $f_i = M(e_i)$. For an empty path, the identity function is the path function.

The path function computes the realizable paths of an arbitrary activity pair. If the path function is applied to a path starting with a producer activity and ending with a consumer activity, the path function returns all realizable non-compliant paths in the process map. In other words, the path function is used to solve the taint flow analysis problem.

Definition 4.5 (Meet-over-all-valid-paths solution)**.** Let $TFA = (P^*, D, F, M)$ be a taint flow analysis problem instance. The *meet-over-all-valid-paths* solution of TFA is a collection of values MVP_n defined as follows:

$$MVP_n = \bigcup_{q \in paths(s_p, n)} pf_q(\top) \quad \text{for each } n \in N^* \tag{4.2}$$

where $paths(m, n)$ is the function that returns all possible paths between m and n in the given process map.

The solution of the taint flow analysis problem is defined as the meet-over-all-valid-paths solution that is the set of realizable paths between activities calculated for each node in the process map. All compliance checks $d \in D$ that are potentially violated at activity n are contained in the set MVP_n, i. e., there exists a realizable path between s_p and n for every compliance check d. We extend the violation criterion by saying that a compliance check d is potentially violated only if n is a consumer activity of d.

4.3.2 Distributive Functions

Paths of the exploded supergraph are maintained by distributive functions $f : 2^D \rightarrow 2^D$. They determine when a non-compliant path is produced, sanitized, or forwarded. Distributive functions are defined over activity transitions in the process map and return, given a set of input paths, the set of output paths.

In order to efficiently compute the result of the taint flow analysis, a special non-compliant path is maintained: the **o**-path, also called *taint path*. The main idea is that a tainted path in the exploded supergraph can only be produced by deriving a path from an already tainted path, i. e., a non-compliant path. For being able to generate a tainted path from each node of the process map, the special **o**-path is always forwarded. A taint path can only be generated by adding an edge from the **o**-path. This is the case when the distributive function generates a non-compliant path for a producer activity. Instead of searching for realizable paths between activities, the problem is simplified: If there exists a realizable path between the start activity and the consumer activity, i. e., a non-compliant path exists at the consumer activity, the corresponding compliance check is violated.

This simplification can be achieved by introducing a compact representation of the distributive function (see Definition 4.6).

Definition 4.6 (Compact representation of a distributive function)**.** The representation relation $R_f \subseteq (D \cup \{\mathbf{o}\}) \times (D \cup \{\mathbf{o}\})$ for a function f is defined as follows:

$$
\begin{aligned}
R_f =_{df} & \{(\mathbf{o}, \mathbf{o})\} \\
& \cup \{(\mathbf{o}, y) : y \in f(\mathbf{o})\} \\
& \cup \{(x, y) : y \in f(\{x\}) \text{ and } y \notin f(\mathbf{o})\}
\end{aligned}
$$

Let f be a distributive function, then R_f is the corresponding compact representation of f. R_f is a graph with $2 \cdot (|D| + 1)$ nodes where each node represents a compliance check $d \in D$ and the **o**-nodes.

Similar to the concatenation of distributive functions $f \circ g$, Reps et al. [RHS95] prove that the concatenation also applies to the corresponding relational representation $R_f ; R_g$.

A distributive function modifies a path between nodes $n_{src} \in N^*$ and $n_{dest} \in N^*$ by applying three different types of operations:

FORWARD A path is forwarded for all transitions in the process map when no producer, sanitizer, or consumer activity is involved. For each compliance check $d \in D$ the path is forwarded, which leads to the following path in the exploded supergraph:

$$\langle n_{src}, d \rangle \rightarrow \langle n_{dest}, d \rangle$$

PRODUCE A path is produced for a compliance check $d \in D$ when a producer activity is part of the transition. In this case, a non-compliant path is derived from the **0**-path because only paths from the tainted path are considered as non-compliant. In the exploded supergraph, the following path is generated:

$$\langle n_{src}, \mathbf{0} \rangle \rightarrow \langle n_{dest}, d \rangle$$

SANITIZE A non-compliant path is sanitized when a sanitizer activity is involved in the transition of the process map. Given a compliance check, no path is forwarded or produced in the exploded supergraph.

Example 4.4 shows how the three operations influence the paths of the exploded supergraph visually.

Example 4.4

Consider two compliance checks, a and b. Figure 4.3 shows three examples that demonstrate the three different types of operations that can be applied to non-compliant paths. The first example shows the forward operation for two compliance checks, where the distributive function f behaves like the identity function.

For the second example, a taint flow is produced for the compliance check a; the other non-compliant path b is forwarded. In this case, the distributive function returns the **0**-flow and the compliance check a for the **0**-flow input to generate a taint flow. Given the compliance check b, the distributive function forwards the non-compliant path for b.

The last example shows how a non-compliant path is sanitized. Here, the distributive function returns the empty set for the corresponding compliance check a and forwards all other non-compliant paths.

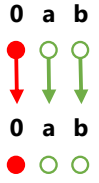
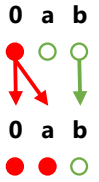
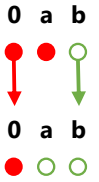
forward(a,b)	produce(a) forward(b)	sanitize(a) forward(b)
$f(\{0\}) = \{\}$ $f(\{a\}) = \{a\}$ $f(\{b\}) = \{b\}$	$f(\{0\}) = \{a\}$ $f(\{a\}) = \{a\}$ $f(\{b\}) = \{b\}$	$f(\{0\}) = \{\}$ $f(\{a\}) = \{\}$ $f(\{b\}) = \{b\}$
$R_f = \{(0,0), (a,a), (b,b)\}$	$R_f = \{(0,0), (0,a), (b,b)\}$	$R_f = \{(0,0), (b,b)\}$
		

Figure 4.3: Example paths for producer, sanitizer, and other activities represented as a graph. Taint paths are marked as red with a closed circle and non-taint paths as green with an open circle. [See+16a].

The three path operations are modeled as a single distributive function that maintains paths for a given set of compliance checks. This function is defined over a transition $(n_{src}, n_{dest}) \in E^*$:

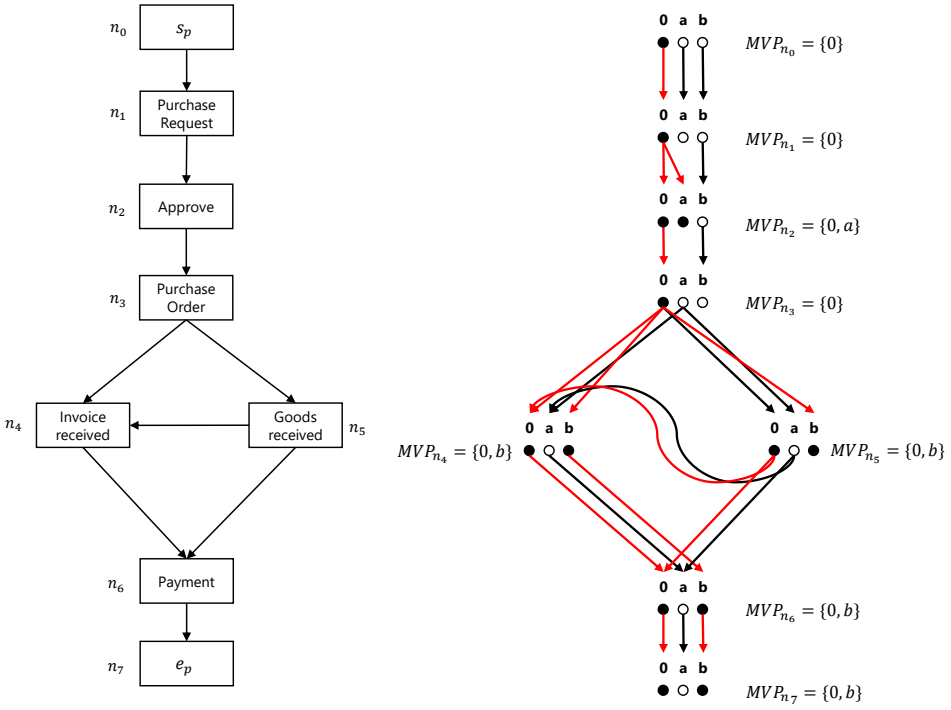
$$f_{(n_{src},n_{dest})}(x) = \begin{cases} \{d : d \in producers(n_{src})\} & \text{if } x = \mathbf{0} \\ \emptyset & \text{if } x \in sanitizers(n_{src}) \\ \{x\} & \text{otherwise} \end{cases} \quad (4.3)$$

The functions $producers(n)$ and $sanitizers(n)$ return a set of compliance checks that contain the activity $n \in N^*$ as a producer or as a sanitizer activity. The distributive function f is defined over all transitions $e_{n_{src},n_{dest}} = (n_{src}, n_{dest})$ in the process map, so it can be used for all transitions in the TFA problem instance as $M(e_{n_{src},n_{dest}}) = f_{(n_{src},n_{dest})}$.

The resulting exploded supergraph maintains all non-compliant paths for all compliance checks and transitions in the process map. In particular, the edges of the exploded supergraph correspond to the representation relations introduced in Definition 4.6 on the transitions of the process map P. Each transition appears $|C| + 1$-times in the exploded supergraph to maintain the non-compliant path for each compliance check and the **0**-flow.

Definition 4.7 (Exploded Supergraph)**.** Let $TFA = (P, D, F, M)$ a taint flow analysis problem instance, the corresponding exploded supergraph $P^\#$ is defined as follows:

$P^\# = (N^\#, E^\#)$, where



(a) the process graph representation of the process

(b) the corresponding supergraph of the process graph

Figure 4.4: An example procurement handling process represented as a process map (a) and the corresponding exploded supergraph (b). The example shows two compliance checks a producer: n_1, sanitizer: n_2, consumer: n_3; b producer: n_3, sanitizer: n_5, consumer: n_6. Open dots indicate that a path is not reachable from $\langle s_p, \mathbf{0} \rangle$ whereas closed dots indicate that there exists a realizable path from $\langle s_p, \mathbf{0} \rangle$, i. e., it is non-compliant. [See+16a].

$$N^{\#} = N^* \times (D \cup \{\mathbf{0}\})$$

$$E^{\#} = \{\langle m, d_1 \rangle \to \langle n, d_2 \rangle : (m, n) \in E^* \wedge (d_1, d_2) \in R_{M(m,n)}\}$$

The exploded supergraph contains all activities of the original process map but they are of the form $\langle n, d \rangle$, where n is the activity in P, and d the compliance check. Each edge of the process map is exploded into multiple edges of the form $\langle m, d_1 \rangle \to \langle n, d_2 \rangle$, where each exploded edge corresponds to a non-compliant path (d_1, d_2) of the compliance check d. A compliance check is potentially violated if the exploded supergraph contains a realizable path $\langle s_p, \mathbf{0} \rangle \to \langle n, d \rangle$ (see Theorem 3.8 in [RHS95]), where n is a consumer activity of d.

Example 4.5

Figure 4.4 shows how the corresponding exploded supergraph (b) is defined for a given process map (a). The process map is generated from an event log as defined in Definition 2.9, including the start and end

node. A closed black circle indicates a reachable node in the exploded supergraph, i. e., there exists a realizable path from $\langle s_p, \mathbf{0} \rangle$. An open black circle indicates a non-reachable node. The example evaluates two different compliance checks.

The first compliance check a evaluates if, after each execution of the *Purchase Request* activity, the *Approved* activity is performed before the *Purchase Order* is executed. The process map illustrates that there is no realizable path such that *Purchase Order* is executed without executing *Approve*. In this case, the compliance check a is fulfilled. The non-compliant path that is produced on the edge $\langle n_1, \mathbf{0} \rangle \rightarrow \langle n_2, a \rangle$ is immediately sanitized at the transition $n_2 \rightarrow n_3$. This results in the meet-over-all-valid-paths solution of $MVP_{n_3} = \{\mathbf{0}\}$. The compliance check is not violated, because the set does not contain a as the corresponding compliance check.

The second compliance check b evaluates if after the execution of *Purchase Order* the *Goods Received* activity is executed before the *Payment* is performed. *Purchase Order* (n_3) is a producer activity which is followed by two other activities n_4, n_5. In both cases a non-compliant path is produced: $\langle n_3, \mathbf{0} \rangle \rightarrow \langle n_4, b \rangle, \langle n_3, \mathbf{0} \rangle \rightarrow \langle n_5, b \rangle$. The non-compliant path is destroyed between the transitions $n_5 \rightarrow n_4$ and $n_5 \rightarrow n_6$, because *Goods received* is the sanitizer activity of compliance check b. However, the non-compliant path is forwarded $\langle n_4, b \rangle \rightarrow \langle n_6, b \rangle$ because $MVP_{n_4} = \{\mathbf{0}, b\}$. In particular, the compliance check is potentially violated because n_6 is a consumer activity of compliance check b, and there exists a realizable path between $\langle s_p, \mathbf{0} \rangle$ and $\langle n_6, b \rangle$ which also corresponds to the result of the taint flow analysis $MVP_{n_6} = \{\mathbf{0}, b\}$.

This section introduced the formal definition of compliance checking using taint flow analysis. We want to show that the correctness proof introduced by Reps et al. [RHS95] can also be applied to our approach.

Similar to the original method, we construct an exploded supergraph for which we investigate realizable paths. Our exploded supergraph is constructed such that edges and nodes correspond to transitions and activities in the given process map. Edges are added to the exploded supergraph according to a customized distributive function that searches the process map for sensitive producer and consumer activities. The distributed function creates and forwards paths whenever the transition of the process map influences the corresponding compliance check. Conformance-establishing activities mark a given path as compliant, i. e., sanitizing a non-compliant path. Each path through the exploded supergraph that is derived from the 0-path is a realizable non-compliant path that can be executed according to the process map. As a result, checking if there is a realizable path between a producer

Algorithm 2: Tabulation algorithm

```
1  Function Tabulate(P#):
2      let (N#, E#) = P#
3      PathEdge ← {⟨s_p, 0⟩ → ⟨s_p, 0⟩}
4      WorkList ← {⟨s_p, 0⟩ → ⟨s_p, 0⟩}
5      ForwardTabulateSLRPs()
6      for n ∈ N* do
7          X_n ← {d_2 ∈ D : ∃d_1 ∈ (D ∪ {0}) such that ⟨s_p, d_1⟩ → ⟨n, d_2⟩ ∈
             PathEdge}
8      end
9  Function Propagate(e):
10     if e ∉ PathEdge then
11         insert e into PathEdge
12         insert e into WorkList
13     end
14 Function ForwardTabulateSLRPs():
15     while WorkList ≠ ∅ do
16         select and remove an edge ⟨n_1, d_1⟩ → ⟨n_2, d_2⟩ from WorkList
17         for ⟨m, d_3⟩ such that ⟨n_2, d_2⟩ → ⟨m, d_3⟩ ∈ E# do
18             Propagate(⟨s_p, d_1⟩ → ⟨m, d_3⟩)
19         end
20     end
```

and a consumer activity without a conformance-establishing activity in the process map corresponds to the reachability of a consumer activity in the exploded supergraph. Due to this construction, the same correctness proof by Reps et al. is also applicable to our compliance checking method.

4.3.3 Tabulation Algorithm

In this section, we introduce our modified version of the tabulation algorithm to efficiently calculate the solution of the taint flow analysis problem.

4.3.3.1 Worklist Algorithm

The tabulation algorithm is a work-list algorithm that maintains a set of all existing *PathEdges* in the exploded supergraph $P^{\#}$ of the process map P. A *PathEdge* is a realizable path in the process map. The tabulation algorithm only manages *PathEdges* of realizable paths from the start node s_p for efficient computation. Specifically, all *PathEdges* are of the form $\langle s_p, \mathbf{0} \rangle \rightarrow \langle n, d \rangle$, where n is a corresponding node in the process map P, and d a compliance check or the 0-path.

The algorithm is illustrated in *Algorithm 2*. The *WorkList* is initialized with the *PathEdge* $\langle s_p, \mathbf{0} \rangle \rightarrow \langle s_p, \mathbf{0} \rangle$ in Line 4. Realizable paths are stored in the

PathEdge set. The main algorithm is performed by the *ForwardTabulateSLRP* function which consists of a loop that is executed as long as the *WorkList* contains items (Lines 14 - 20). The *WorkList* only contains *PathEdges* that have not yet been visited by the algorithm. This condition allows the algorithm to terminate, even for process maps that contain loops. The tabulation algorithm propagates a new *PathEdge* $\langle s_p, \mathbf{0} \rangle \rightarrow \langle m, d_3 \rangle$ whenever there is an edge of the form $\langle n_2, d_2 \rangle \rightarrow \langle m, d_3 \rangle \in E^{\#}$ (Lines 17 - 19). In order to determine if there exists an edge, the actual implementation evaluates the distributive function $f_{(n_2,m)}(d)$, which returns non-compliant paths of the transition between n_2 and m for a compliance check d.

As a result, the tabulation algorithm constructs a set of potentially violated compliance checks $d \in D \cup \{\mathbf{0}\}$ for each node n (Lines 6 - 8). A compliance check is potentially violated if there exists a realizable path for d to node n, which is equivalent to a *PathEdge* $\langle s_p, \mathbf{0} \rangle \rightarrow \langle d, n \rangle$. The algorithm returns multiple sets X_n that contain the compliance checks for which a non-compliant path from the tainted start node $\langle s_p, \mathbf{0} \rangle$ exists. A potential violation is detected when node n is a consumer activity of one of the compliance checks returned by X_n.

4.3.3.2 Computational Complexity

The cost of the tabulation algorithm depends on the number of transitions and the number of compliance checks including the 0-path that should be evaluated. The output of the distributive function is calculated for each transition of the process map to determine if a non-compliant path is produced, forwarded, or sanitized. For the specific case of checking compliance on the control flow perspective, the output of the distributive function only affects a single compliance check. Thus, it at most creates a single edge in the exploded supergraph to be traversed. Typically, most transitions are not influencing the output of the distributive function at all, which equals to the identity function. As a result, evaluating the distributive function does not add more than one edge that needs to be visited by the tabulation algorithm. Under the assumption that all primitive operations have the same cost, the runtime of the tabulation algorithm is $O(|E| \cdot |D|)$, where $|E|$ is the number of transitions, and $|D|$ the number of compliance checks including the $\mathbf{0}$-path.

4.3.4 Replay of Process Variants on Exploded Supergraph

The final step checks each process variant of the event log if it follows a violated path in the exploded supergraph or not. The algorithm replays the corresponding trace on the taint flow analysis result because the process map allows more behavior than what was observed in the event log. Each process variant is replayed on the taint flow analysis result to check if there exists a realizable path from a producer activity to a consumer activity. The replay

of the process variants is efficient because the realizable paths have already been calculated by the tabulation algorithm.

Each process variant is encoded as a sequence of activities

$$v_i = \langle n_0, n_1, ..., n_{m-1}, n_m \rangle$$

such that the corresponding producer and consumer activities for each compliance check $d \in D$ are identified. Let $n_i \in producers(d)$ and $n_j \in consumers(d)$ in the sequence v_i, such that

$$v_i = \langle n_0, n_1, ..., n_i, n_{i+1}, ..., n_{j-1}, n_j, ..., n_m \rangle$$

If producer and consumer activities are contained in the activity sequence and all nodes in between $n_{i+1}, ..., n_{j-1}, n_j$ are part of a taint flow, the process variant violates against the compliance check:

$$\forall n \in \langle n_{i+1}, ..., n_{j-1}, n_j \rangle : d \in MVP_n$$

As a result, the algorithm returns a list of affected process variants for which the compliance checks fail.

4.4 Evaluation

This section presents the results of an experimental evaluation, investigating the applicability and performance of the TFA compliance checking algorithm.

4.4.1 Experiment Setup

The TFA algorithm is implemented as a ProM[1] 6 [Ver+11] framework plugin. For the evaluation, we use a set of real-life and synthetic event logs. A set of compliance rules is defined to check the compliance of event logs.

4.4.1.1 Event Logs

Two real-life event logs from SAP ERP systems are used to evaluate the applicability and performance of our algorithm. Both event logs contain information about the execution of a procurement handling process. Furthermore, synthetic event logs from a manually crafted BPMN model are generated using PLG2 [BS11]. PLG2 is a tool that allows generating event logs from process models by simulating the execution of the process model. For the synthetic event logs, the probabilities for *trace missing head*, *trace missing tail*, *trace missing episode*, *perturbed event order*, *doubled events*, and *alien events* are set to 20% to obtain event logs with a certain amount of variation.

1 ProM is an open-source tool for process mining, developed by the Eindhoven University and used in the academic field to quickly develop and explore new methods. It is highly extensible via plugin capabilities.

	Event Log A	Event Log B	Event Logs C
Type	SAP ERP	SAP ERP	Synthetic
Period	1 month	2 years	-
Cases	873	651 709	1000 - 1 000 000
Events	5 077	3 880 138	varying
Event classes	20	35	15

Table 4.2: General properties of the event logs used in the evaluation.

All three event logs represent the same type of procurement handling process with the same set of activities. A discovered causal-net from event log A is shown in Figure 4.5. The underlying procurement handling process works as follows: First, a shopping cart of items is created which is then approved by the department manager. Each approved shopping cart leads to the actual order of the requested items. After the purchased items are delivered, the invoice is received which is then paid. Various non-compliant activities can be executed although this process seems quite simple.

Table 4.2 shows a summary of the properties of the event logs used. Event log A only consists of a few cases and events from a small period of one month. Event log B consists of events collected over two years from a single organization. The synthetic event logs (event logs C) vary in the number of cases between $1,000$ and $1,000,000$ to evaluate the performance concerning the size of the event log.

4.4.1.2 Compliance Checks

A set of four compliance checks that refer to the procurement handling process is defined for all three event logs introduced above. These compliance checks are designed based on compliance guidelines that organizations typically have to follow, and are defined as follows:

1. **Approval of all Purchase Requests.** Any free-text purchase request of a department needs to be approved before the requested items are ordered.
2. **Approval of all Shopping Carts.** Alongside free-text purchase requests, departments can order from a set of items in a catalog like at an internal "Amazon". Similar to purchase requests, these shopping carts also need to be approved by a manager.
3. **Approval of all Purchase Orders.** The procurement department also needs to approve purchase orders after the purchase request or the shopping cart is approved by the department manager.
4. **Goods must be received before Payment.** Any good that is ordered must be received at some time before the payment is executed.

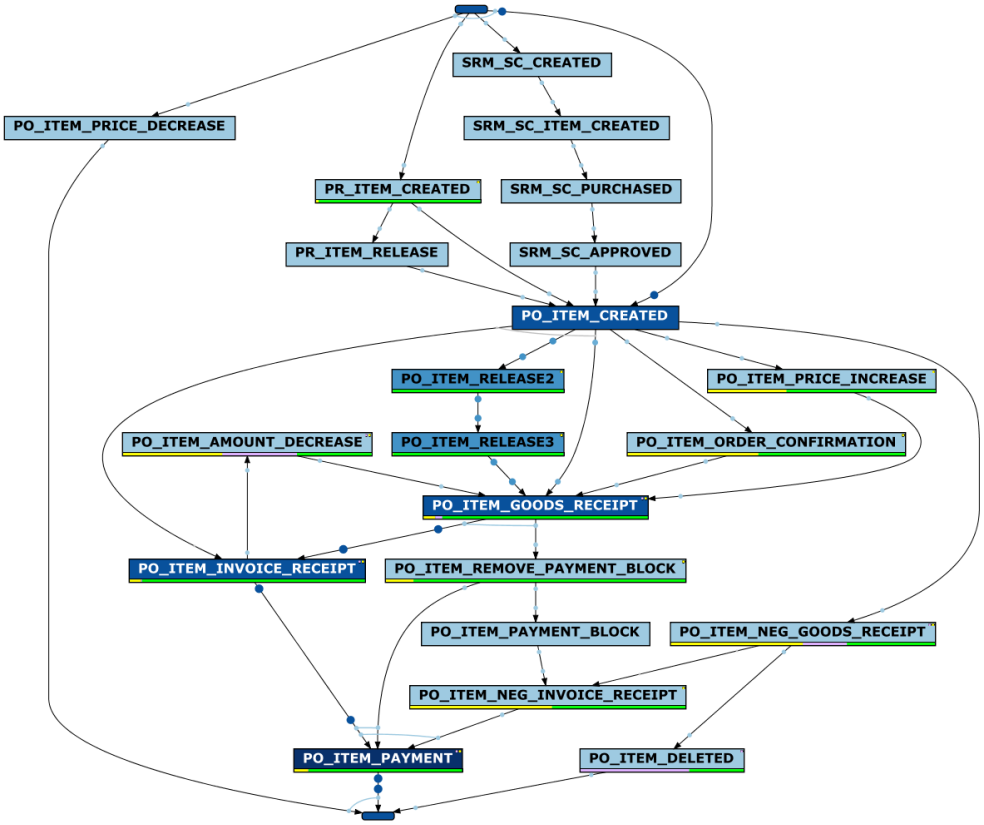


Figure 4.5: Causal net discovered with the heuristics miner of ProM showing the event log A.

The four compliance checks are evaluated for the three different event logs using the taint flow analysis algorithm, the LTL checker [ABD05], and the Petri net pattern matcher [RFA12]. For each approach, the corresponding rule representation is generated. Example 4.4.1.2 illustrates the generation of the compliance check rules.

Example 4.6

In this example, we consider the "Approval of all Purchase Requests" rule introduced above. It is transformed into a 3-tuple of activities as follows:

$$CC = (\overbrace{\{\text{Purchase Request}\}}^{\text{producer}}, \overbrace{\{\text{Request Approval}\}}^{\text{sanitizer}}, \overbrace{\{\text{Purchase Order}\}}^{\text{consumer}})$$

The same rule is transformed into an LTL expression used for the LTL checker and the Petri net pattern matcher as follows:

$$\begin{aligned}&[]((\text{task} == \text{"Purchase Request"}\\ &\quad -> (\text{task!} = \text{"Purchase Order"}\\ &\quad \_U\ \text{task} == \text{"Request Approval"})))\end{aligned}$$

4.4.1.3 Hardware Specifications

The performance benchmarks were performed on a computing server machine. The server machine was equipped with two Intel Xeon E5-2640 v2 processors, each contains 8 cores with 16 threads, and 64 GB of memory. Ubuntu 16.04.4 LTS was installed on the machine along with OpenJDK 1.8.0_162. The developed ProM plugin was run on ProM 6.8 that comes with the version for the LTL checker and the Petri net pattern matcher plugin. Table 4.3 shows the specifications of the used machine.

	Compute Server
Processor	2x Intel Xeon E5-2640 v2
Memory	64 GB
Storage	800GB SSHD
OS	Ubuntu 16.04.4 LTS
Java VM	Oracle OpenJDK 1.8.0_162

Table 4.3: Compute server specifications.

4.4.2 Measurement Method

For the experimental evaluation, two measures were evaluated to assess the applicability and computational performance.

4.4.2.1 Number of Compliance Violations

The number of compliance violations was measured for the real-life and the synthetic event logs to show the applicability. The four compliance checks were validated against the event logs, and we measured the number of compliant and non-compliant cases.

4.4.2.2 Runtime

The total runtime to execute the compliance checking algorithms until retrieving the violations was measured. All the required steps to check the compliance of an event log were measured, including the conversion of the event log into a process map, the execution of the tabulation algorithm, and the replay of the process variants on the exploded supergraph. For the LTL Checker and the Petri net pattern matcher, the computational time for the actual rule checking was considered. All other computation times, such as loading the event log as well as the visualization, were excluded. Each performance benchmark was executed six times for each event log, ignoring the first measurement due to the Java VM overhead for memory allocation.

Dataset	Rule	Ground Truth	Taint Flow	LTL Checker	Petri net pattern
Event Log A	(1)	65	65	65	65
	(2)	0	0	0	0
	(3)	342	342	342 (346)	342
	(4)	3	3	3 (5)	3
Event Log B	(1)	293 588	293 588	293 588	293 588
	(2)	0	0	0	0
	(3)	323 539	323 539	323 539	323 539
	(4)	2 679	2 679	2 679	2 679
Event Log C	(1)	24	24	24 (54)	24
	(2)	24	24	24 (52)	24
	(3)	52	52	52	52
	(4)	155	155	155	155

Table 4.4: Number of cases that violate against a compliance check.

4.4.3 Results

This section presents the results of the experimental evaluation. A comparison with the LTL Checker and the Petri net pattern matcher shows the differences between the approaches.

4.4.3.1 Compliance Violations

The first part of the experimental evaluation shows the results of the detected compliance violations.

We found deviations between the ground truth labels of the event logs and the results of the compliance checkers. The LTL checker identified different cases as violating, although they were not tagged as such. For event log A and rule (3) there are 4 cases where no purchase order is performed. However, the LTL Checker considers these as violations. Similar, two cases for rule (4) are marked as violated by the LTL Checker because no payment was executed at all. It turns out that the LTL expression is not semantically the same as the compliance checks of our approach. The same issue is also present in the synthetic event log, where traces are marked as violated although these cases never executed the triggering producer activity. Table 4.4 shows the compliance rule violations for each of the defined compliance rules and event logs.

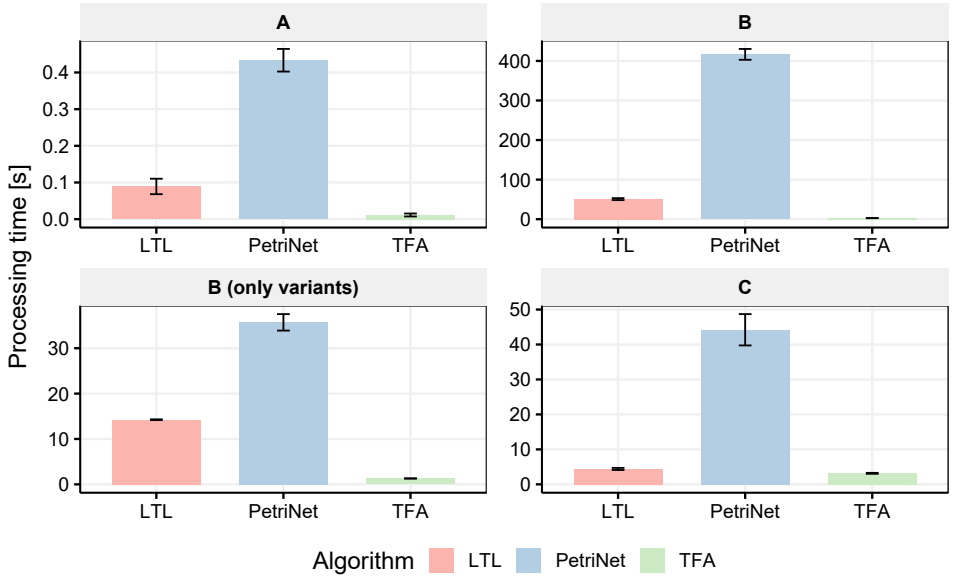


Figure 4.6: Average runtime of the compliance checking approaches over a total number of 5 runs.

4.4.3.2 Performance Benchmark Results

Figure 4.6 shows the average total computation time grouped by the different approaches and event logs over 5 different runs.

We perform statistical significance testing to show that our approach outperforms the state of the art methods regarding performance. A non-parametric Friedman test [Fri37] showed that the compared approaches perform significantly[2] different. Post-hoc analysis based on Wilcoxon signed-rank tests and a Bonferroni correction shows that the taint flow analysis is statistically significantly faster compared to all other methods. In particular, the deviation becomes more visible for large event logs. This is not surprising because the related work implementations do not group cases with the same sequence of events together, but evaluate each case individually. For a fair comparison, the event log B was modified to contain unique cases only. From the results, the taint flow analysis algorithm is still able to check the compliance of the predefined rules quicker.

Table 4.5 presents the performance results of the compared methods. A detailed inspection of the benchmark results shows that the taint flow analysis algorithm performs, on average, 7.7-times faster compared to the LTL Checker for event log B and about 7.8-times faster for the event log A. For the synthetic event log C, our algorithm performs 1.4-times faster on average across all configurations. We found that the performance difference increases with the increasing number of cases within an event log. The Petri net pattern

2 Non-parametric Friedman test [Fri37]; $\chi^2(2) = 40$, $p < 0.001$

	Log A	Log B	Log B (only variants)	Log C
Petri net Replay	433.4	415 602.6	35 707.0	44 214.0
LTL	89.2	50 666.2	14 244.3	4 401.2
TaintFlow	**11.4**	**6 616.2**	**1 294.0**	**3 150.8**

Table 4.5: Performance Benchmark (in milliseconds) for Taint Flow Analysis, LTL Checker, and Petri net Replay.

matching approach performs slowly on large event logs. The adjustment that is made to each trace to fit the predefined patterns may significantly slow down the computation. Unlike the other two methods, the computation time of the taint flow analysis algorithm mainly depends on the number of compliance checks and the number of transitions in the process map. The advantage of our approach is that the precomputed exploded supergraph can be reused for each trace which significantly reduces computation time.

4.4.3.3 Influence of Event Log Size on Runtime Performance

The results of the two real-life event logs considered reveal that the number of traces in an event log influences the performance of the algorithms. In this experiment, the size of the synthetic event log is varied between 1,000 and 1,000,000 traces to get a detailed benchmark of the performance between the compared methods.

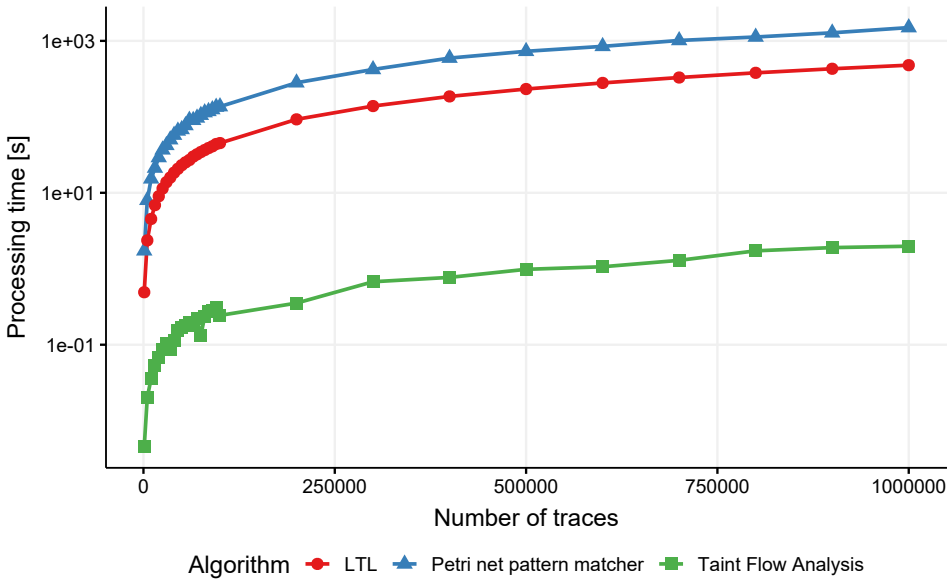


Figure 4.7: Processing time in seconds for the synthetic event logs depending on the number of traces in the log.

The results reveal that the TFA algorithm outperforms the two other methods concerning runtime performance. Specifically for large event logs, the number of traces has a significant influence on the processing time of all methods. Figure 4.7 reports the processing time across all runs depending on the number of traces in the event log. Note that the figure shows an aggregated view across all different runs, including the variation in the number of rules and that the y-axis is log-scaled.

4.4.3.4 Influence of Number of Rules on Runtime Performance

According to the complexity analysis in Section 4.3.3.2, the runtime performance of our approach is influenced by the number of rules. The exploded supergraph maintains as many paths as compliance checks. The LTL checker and the Petri net pattern matcher generate specific models for each of the compliance rules. In this experiment, the number of rules is varied between 5 and 50 to get a detailed benchmark of the performance between the compared methods.

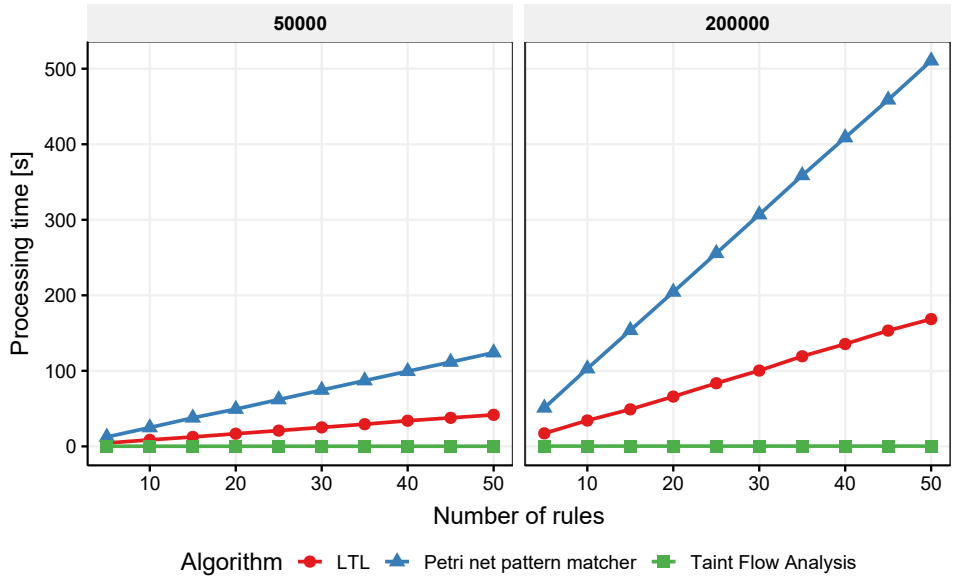


Figure 4.8: Processing time in seconds for the synthetic event logs (log size=50 000 and 200 000) depending on the number of rules in the log.

Figure 4.8 reports the processing time over two selected event log sizes and the number of rules. The results indicate that the LTL checker performance depends on the number of rules to check. For the Petri net pattern matcher, a linear increase of processing time can be observed which is intuitive because for each rule a separate Petri net is generated.

4.5 Discussion and Limitations

We presented a backward compliance checking algorithm based on the graph-reachability problem. However, there are some limitations and research directions for future work.

4.5.1 Compliance Rules

The compliance rule definition is limited to a three-tuple of producer, sanitizer, and consumer activities. This notation is sufficient to describe a considerable subset of the practically relevant control flow constraints, such as the presence of an activity or the order of activities. However, complex dependencies beyond activity co-occurrence and eventually/directly followed relationships cannot be described with our simplified notation. Cardinal-based rules, data-driven rules, related time rules, or static property rules are not supported out-of-the-box. For instance, our approach also does not support the direct inclusion of other process perspective such as the organizational perspective. Slight modifications on how the process map is generated may overcome some of these limitations. For example, checking the segregation of duty rule or the four-eyes principle is possible if the concatenation of activity and resource name of an event is used as the classifier function. Further research how other complex business rules can be checked using our approach is necessary.

4.5.2 Process Map

Instead of using process maps as the graph representation, other more expressive process model representations, such as Petri nets or BPMN may be used to build the exploded supergraph. Such models may be discovered by process mining discovery algorithms with a high process model fitness. Accurate and precise models may be used to avoid the replay of traces. However, using different process model representations requires various modifications to the tabulation algorithm to support additional semantics such as concurrency.

4.5.3 Counter-example

Another research direction for future work is to enhance the visual explanation of counter-examples. In the current approach, only one counter-example is presented to the analyst by highlighting one realizable path in the process map. Further improvements such as highlighting all possible realizable paths as well as how many cases are affected may be needed. This requires that the replay of the traces need to be connected to the realizable paths identified by the taint flow analysis.

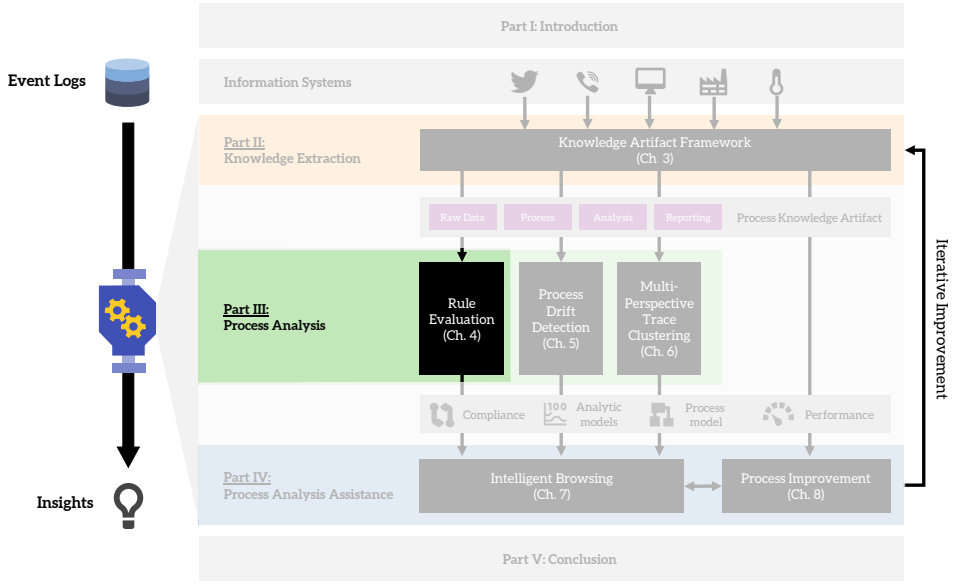


Figure 4.9: Overview of the chapter and contributions.

4.6 Conclusion

In this chapter, we presented a novel business rule evaluation algorithm that transforms the problem of backward compliance checking into a graph-reachability problem. In summary, the main contributions of this chapter are:

1. A fast process compliance checking algorithm that identifies compliance violations concerning the existence and the order of activities in event logs.
2. The transformation of backward compliance checking into a graph-reachability problem which can be efficiently solved using a modified version of the dataflow analysis framework [RHS95].
3. A method to provide a visual explanation of the violated compliance check, returning a realizable path in the process map.
4. Results of an experimental evaluation investigating the applicability and computational performance of the TFA algorithm.

The backward compliance checking algorithm introduced in this chapter identifies compliance violations concerning the existence and the order of activities in event logs. Instead of using a model-based compliance checking method, the problem is formulated as a graph-reachability problem. As illustrated in Figure 4.9, backward compliance checking contributes to the overall thesis goal of obtaining valuable insights about a business process from event logs. Particularly, it allows analysts to check for typical process problems concerning compliance issues on the control flow perspective. In

the next chapter, we introduce a more autonomous algorithm to identify process behavior changes over time.

5

Process Drift Detection

The previous chapter presented a targeted analysis task, validating business rules in historic event logs. This chapter introduces a novel process analysis algorithm that automatically detects process drifts in event logs to improve process discovery. In particular, it identifies time periods when a business process has changed its execution behavior and provides characterizations of the process drifts detected. The algorithm explores subsets of consecutive cases in an event log by discovering multiple process models and analyzing their structural properties. The results of the algorithm can be visually highlighted in the corresponding process model. In contrast to the related work, our algorithm is based exclusively on event logs, recognizes all different process drift types, is resistant to noisy event logs, provides process drift characterizations, and is parameter-free.

This chapter is organized as follows. In Section 5.1, a motivating introduction to process drift detection is given. Next, related work in the field is introduced (Section 5.2). Section 5.3 then presents a novel process drift detection algorithm. Then, Section 5.4 reports the results of the experimental evaluation. The chapter concludes with the limitations of the algorithm, and future work (Section 5.5).

> **Publication:** This chapter is based on the following publication:
>
> Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. "Detecting Concept Drift in Processes using Graph Metrics on Process Graphs." In: *Proceedings of the 9th Conference on Subject-oriented Business Process Management - S-BPM ONE '17*. ACM Press, 2017. DOI: 10.1145/3040565.3040566.
>
> **Contribution Statement:** I led the idea generation, implemented the prototype and performed the data evaluation. *Timo Nolle* and *Max Mühlhäuser* supported the conceptual design and contributed to the writing process.

5.1 Introduction and Motivation

Business processes are constantly changing due to various reasons. On the one hand, a major part of these changes are caused by external forces such as supply and demand variations, seasonal shifts, or new customer needs [Bos+11; Bos+14; Maa+15; Ost+16]. Organizations continuously deploy modifications to their process with the goal of improving their execution and increasing the return on investment. On the other hand, new governmental regulations such as the *Sarbanes-Oxley Act* [SO02] or the General Data Protection Regulation (GDPR) [Tan16] force organizations to change their processes. Consequently, these changes lead to a different execution of the corresponding business process, also called *process drift*. A process drift is a significant behavior change of the execution of a process that occurs over time. The change can be *planned and documented*, this is the case when the organization has intended to apply a desired change to the process, or *unexpected*, this is the case when the change is not intended [Maa+15]. Unexpected changes may be introduced silently without the organization being aware of it.

In addition to the recognized necessity of process drift detection in the Process Mining Manifesto (C4) [IEE11], there are two further important reasons for identifying process drifts:

1. Organizations are specifically interested in detecting unexpected and unknown process drifts because they affect process performance or lead to compliance violations.
2. Existing process analysis methods assume stable processes that can lead to incorrect conclusions [Bos+11]. Frequent changes to the process are also reflected in an event log. These changes may lead to unexpected results when applying process discovery to different periods.

Overall, process drift detection is valuable for organizations to identify hidden process behavior changes as well as it helps to improve the result of other process mining algorithms, such as process discovery.

We distinguish between four different types of process drifts according to Bose et al. [Bos+14], depicted in Figure 5.1:

SUDDEN DRIFT A sudden drift occurs when a process is substituted by another process and there is no overlap between the two processes. Specifically, there exists a definite point in time when process behavior has switched.

For example, sudden drifts often occur in emergency situations where different procedures are applied immediately.

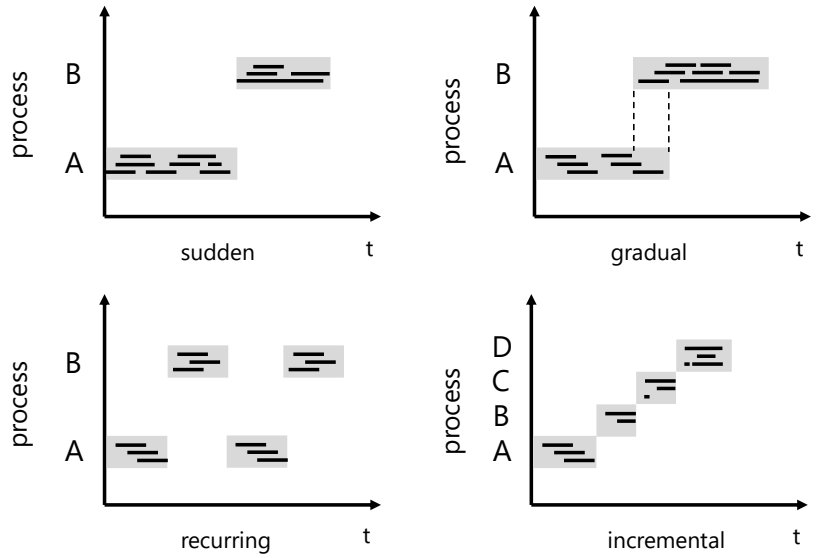


Figure 5.1: Overview of the different process drift types. The x-axis refers to the time and the y-axis refers to the different process behaviors. Shaded areas: process instances. Adapted from [Bos+14].

GRADUAL DRIFT Similar to a sudden drift, a gradual drift occurs when a process is replaced by a different process. However, both processes coexist for a certain period until the new process gradually takes over.

For example, the introduction of an optimized workflow is rolled out gradually over the departments.

RECURRING DRIFT Different from sudden and gradual drifts, in which a new process is established, the characteristic of recurring drifts is the switch back and forth between processes. Such drifts are typically observed in processes that are influenced by seasonal effects.

For example, organizations offer different products or services in summer than in winter.

INCREMENTAL DRIFT An incremental drift is characterized by a step-wise replacement of the process whereas all other three drift types deal with the change of a process from one to another.

For example, optimizing a business process requires multiple changes in different departments which are rolled out incrementally.

This chapter introduces a process drift detection algorithm that aims at finding sudden, gradual, recurring, and incremental drifts on the control flow perspective of a process, regardless if they are planned and documented or unexpected. The basic idea is to discover multiple process models from subsets of consecutive cases in an event log. As a result, multiple process models are discovered that represent the process behavior over time. The algorithm inspects the characteristics of these process models by computing

various graph metrics. Statistical hypothesis testing is applied over consecutive process models to detect process drifts. We use a custom sliding window algorithm over consecutive cases in the event log to obtain the appropriate number of cases to be considered for each subset. Furthermore, the computed graph metrics provide further insights about the identified process drift, unlike most related work do not provide any characterization.

5.2 Related Work

This section introduces related work in the context of process drift detection. There are three fundamentally different approaches to detect process drifts: (1) approaches using additional information of adaptive process management systems, (2) clustering approaches that use similarity-based metrics over traces, and (3) statistical approaches evaluating statistical significance tests.

We compare the related work based on a set of relevant requirements for designing process drift detection algorithms:

(R1) **Rely exclusively on event logs**
The process drift detection algorithm should only consider data that is stored in an event log. In real-life scenarios, other information about the process and its execution, such as process models or other relevant documentation, is typically not available.

(R2) **Detect all process drift types**
The process drift detection algorithm should not be limited to a specific type of process drift but detect all four types [HBA15]. If only specific process drift types are detected, this would require the analyst to have at least some knowledge about which detection algorithm to choose based on the types of process drifts present in the event log.

(R3) **Resist noisy event logs**
In order to apply the process drift detection to real-life event logs, the algorithm should be resistant to noise because no real-life event log is noise-free. Without proper handling of noisy event logs, the algorithm produces a large number of false positives which should be avoided.

(R4) **Provide drift characterizations**
In case of a detected process drift, the algorithm should provide characterizations that describe what changes were detected [CG12; Bos+14]. These characterizations help analysts to better understand what changes were made to the process execution caused by the drift, and if necessary, take actions to resolve potential process issues.

(R5) **Be parameter-free**
Since this thesis aims for algorithms that work for real-life event logs, it is typically not feasible to set method parameters ideally due to the lack

of sufficient prior knowledge. For this reason, process drift detection algorithms should be parameter-free and adjust themselves to retrieve robust and reliable results.

	Rely exclusively on event logs	Detect all process drift types	Resist noisy event logs	Provide drift characterizations	Be parameter-free
Adaptive Process Management Systems					
Günther et al. [Gün+06]	○	◐	◐	●	◐
Similarity-based Detection					
Lakshmanan et al. [LKD11]	●	○	●	●	○
Luengo and Sepúlveda [LS12]	●	●	●	○	○
Accorsi et al. [AS12]	●	◐	●	◐	○
Hompes et al. [HBA15]	●	●	●	○	○
Statistical-based Detection					
Bose et al. [Bos+11]	●	○	●	◐	○
Weber et al. [WBT11]	●	○	●	○	○
Carmona et al. [CG12]	●	○	●	●	●
Bose et al. [Bos+14]	●	○	●	◐	○
Maaradji et al. [Maa+15]	●	●	●	○	●
Martjushev et al. [MBA15]	●	○	●	◐	○
Kumar et al. [MTA15]	●	○	◐	○	○
Ostovar et al. [Ost+16]	●	●	●	◐	○
Richter and Seidl [RS17]	●	◐	●	○	●

Table 5.1: Property comparison of process drift detection algorithms. ● indicates the fulfillment, ◐ indicates a partial fulfillment, and ○ indicates the non-fulfillment of a requirement.

Table 5.1 compares the relevant research in each group regarding the established requirements. The following sub-sections elaborate on the related work in detail.

5.2.1 Adaptive Process Management Systems

Flexible processes in organizations can be supported by adaptive process management systems [RRD04], which allow the organization to adjust the

execution of a process at runtime. Adaptive process management systems provide explicit information about process changes. Günther et al. [Gün+06; Gün+08] investigate the adaptation of a process performed in Process-aware Information Systems (PAISs). Such adaptive process management systems generate additional information that can be explored using process discovery, enabling to extract the changes to the process model.

Regardless of the rare use of adaptive process management systems in organizations, maintained and accurate process models are required. Furthermore, undocumented and unexpected process drifts cannot be detected with these approaches.

5.2.2 Similarity-based Detection

Using explicit information from the inspection of adaptive process management systems to detect process drifts is often impossible because such information is not available. However, recorded event logs indirectly reflect the behavior change in process executions.

Existing approaches detect process drifts in event logs using cluster analysis. Similar process behavior over time is grouped based on a feature vector that describes the behavior of a trace in a process. Multiple groups indicate the presence of a process drift in the event log. Specifically, temporal proximity between traces allows identifying process drifts [LS12]. Accorsi et al. [AS12] construct a similarity measure between activity pairs by inspecting their relative position within a trace. Whenever their relative position changes over time, the similarity between the activities changes as well; revealing a structural change of the process. The authors propose an interactive approach to visually determine the cluster cuts which indicate the start and the end of a cluster on the time axis. Analysts can visually browse through the computed chart and inspect the identified process drifts. Hompes et al. [HBA15] use the Markov cluster algorithm for process drift detection. Here, clustering is used to distinguish between mainstream and deviating behavior of traces in an event log [HB15]. Markov clustering automatically finds the optimal number of clusters to split the event log into appropriate clusters. For the drift detection algorithm, the underlying assumption is that the number of clusters stays stable when the behavior of the process remains similar. A change to the number of clusters indicates a potential process drift.

Lakshmanan et al. [LKD11] introduce a spectral graph analysis approach operating on sets of process traces. The authors define the graph space that corresponds to the directly-follows relation of activities. For multiple sets of traces, a separate graph is created describing the distances between traces as the cosine similarity of directly-follows relations. Then, the eigenvalues of each graph are computed to compare the different periods.

Similarity-based approaches exclusively rely on the information stored in event logs and use clustering or graph analysis to detect the different process behaviors. In general, these approaches are resistant to noisy event logs and reliably detect potential process drifts of all types. However, all methods require the user to set specific parameter values to work appropriately. In most situations, this is trial-and-error because the actual process drifts are unknown.

5.2.3 Statistical Hypothesis Testing based Detection

Another research stream uses statistical hypothesis testing to detect significant deviations between cases of different periods in the event log.

Bose et al. [Bos+11; Bos+14] extract several global and local features from an event log to capture the dependencies between activities over time. Process drifts are detected by exploring two consecutive fixed windows of cases over the partially sorted event log. For each of the features, a statistical hypothesis test detects significant deviations between two consecutive windows. The approach only finds sudden process drifts whereas other types of drifts are not detected. An extension to this work, which additionally finds gradual drifts, is introduced in [MBA15]. Both approaches require prior knowledge about the process to optimally set the size of the inspected windows. In real-life scenarios this is an unrealistic assumption, causing a try-and-error practice. The window size heavily influences the performance of the detection, and it is not feasible to explore all possible window sizes.

A parameter-free fully automated approach is presented by Maaradji et al. [Maa+15]. Instead of extracting numerous features from traces, the authors use statistical hypothesis testing over the distribution of process runs. The issue of setting an appropriate window size is avoided by using two adaptive sliding windows. A process drift is detected whenever the distribution across runs is statistically different. Runs are defined by the α-concurrency derived from the α-miner [AWM04]. An extension to this method is presented in [Ost+16] which also supports the detection of process drifts in event streams. Using event streams instead of historic event logs allows detecting process drifts even if the process instance is still running. A real-time approach using probabilistic deterministic finite automata is presented in [WBT11]. Other researchers use event correlations [MTA15] or learn internal representations [CG12].

Most of the work only inspects the control flow perspective of processes, but other perspectives are also good indicators for process drift detection. In particular, operation times of activities are of high interest because they inherently affect the performance of the overall process execution. TESSERACT [RS17] uses a rolling average over the completion times of activities to identify temporal process drifts in event streams.

Similar to distance-based approaches, statistical-based process drift detection methods exclusively rely on event logs. These approaches are typically very robust against noisy event logs and provide accurate process drift detection. However, most approaches again need the analyst to set appropriate parameter values to obtain useful results. Furthermore, only a very limited set of approaches aims at detecting all process drift types, and only a few provide explanations as to why a drift is detected.

5.3 Detecting Process Drifts using Graph Metrics

This section introduces a novel process drift detection algorithm. It compares the structure of discovered process models from different subsets of consecutive cases in the event log by inspecting various graph metrics to detect potential process drifts. Our process drift detection algorithm only requires a time-sorted event log L. The algorithm is composed of four steps:

1. The event log is split into two smaller sub-logs containing consecutive subsets of cases (Section 5.3.1).
2. For each of the two sub-logs process models are discovered using an existing process discovery algorithm (Section 5.3.2). For the resulting process models, specific graph metrics are calculated.
3. Statistical significance tests determine if the two consecutive sub-logs differ from each other, detecting a process drift (Section 5.3.3).
4. The differences of the graph metrics are explored to characterize the identified process drift (Section 5.3.4).

These steps are repeatedly executed until the event log is fully scanned. Figure 5.2 shows an overview of the four steps of the process drift detection algorithm.

In the following, each step is described in detail.

5.3.1 Splitting Event Log into Reference and Detection Window

In the first step, the event log is split by cases into two smaller consecutive sub-logs, each of size w. In the following, the first sub-log is called *reference window* R, where $R \subset L$, and the second sub-log is called *detection window* D, where $D \subset L$. Each sub-log consists of complete cases which are not split apart. Both windows are adjacent to each other and not overlapping: $R \cap D = \emptyset$. Combined, both windows build a composite window of size $2w$ cases over the event log split by time, such that $R = \langle c_1, c_2, ..., c_w \rangle$ and $D = \langle c_{w+1}, c_{w+2}, ..., c_{2w} \rangle$ with $c_i \in L$ for $1 \leq i \leq 2w$.

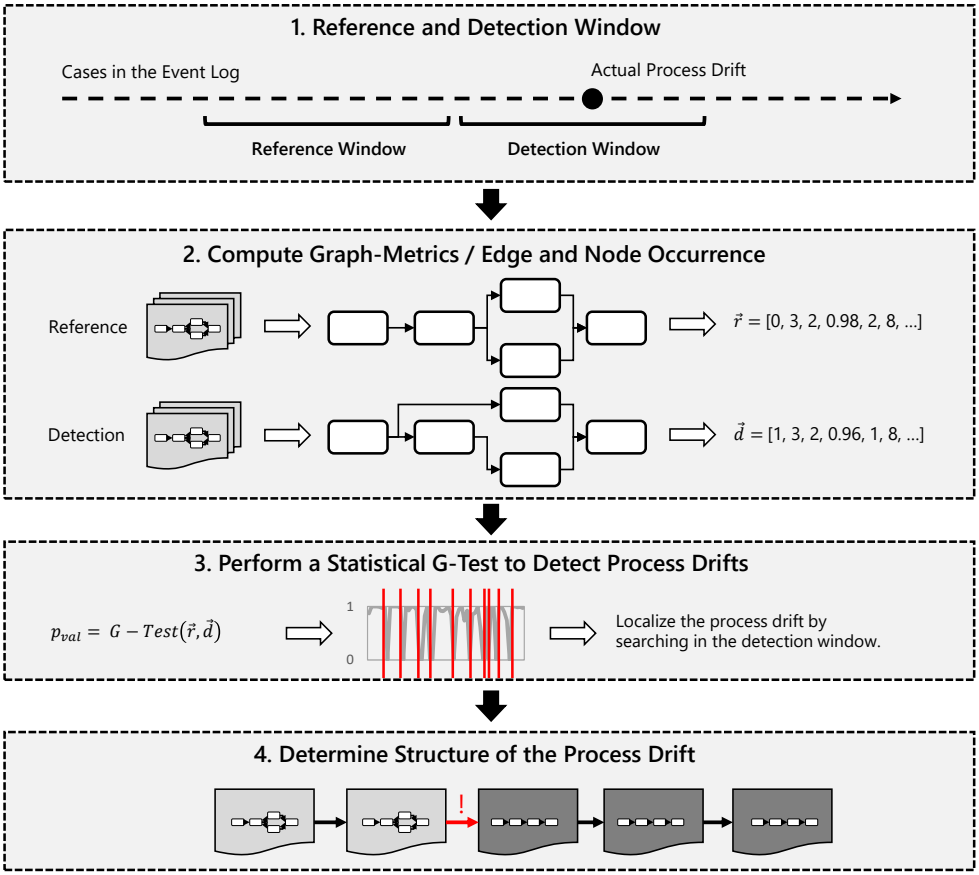


Figure 5.2: Overview of the Process Drift Detection Algorithm.

The fundamental idea of our process drift detection algorithm is to find consecutive sub-logs that significantly deviate from each other, i. e., where cases of the reference and the detection window are different. The algorithm detects a potential process drift if there is a deviation between the two windows. Instead of relying on a fixed window size that needs to be defined based on prior knowledge about the event log, an automated adaptive sliding window approach is used. Algorithm 3 summarizes the adaptive sliding window approach. The algorithm starts with a window size $w = 100$ (Line 1) and increases the size by a factor of 1.2 if no significant difference between the windows is found (Line 23). The algorithm increases the window size up to 200 cases to reduce computational time. If the maximum size is reached, the algorithm moves the reference window to the start of the detection window and resets the window size to $w = 100$ (Line 25 - 28). Here, we assume that there exists no process drift within the inspected composite window, and the algorithm moves to the next set of cases in the event log. However, if a significant difference is detected between the two windows, the algorithm considers a candidate process drift within the composite window.

The accurate position of the candidate process drift is located by repeating significance testing on a smaller window size. Specifically, the considered search space only contains the cases that are in the detection window for which a candidate process drift is detected. The window size is set fixed to half of the current window size (Line 6). Similar to the candidate process drift detection, the smaller window is moved step-by-step over the search space to locate the accurate position of the process drift. The first positive significance test marks the position between the two consecutive windows as the location of the candidate process drift (Line 12 - 16). However, if no significance test is positive, the size of the window is adjusted and moved one window size forward (Line 18 - 21). The algorithm uses an early stopping heuristic (Line 10) to reduce the number of statistical significance tests. The heuristic inspects the probability value of the tests to cancel the process drift search in the given search space.

5.3.2 Compute Process Models and Graph Metrics

In the next step, we mine process models from both sliding windows to determine the deviations of the process execution. The basic idea is that a process drift results in two different process models. We use the Flexible Heuristics Miner (FHM) (see Section 2.4.2) to mine the process models. The advantage of FHM is that it exhibits fast computation and good abstraction of the event logs. It also solves the issues with noisy event logs and concurrent activities. Minor deviations caused by noise in the event log don't lead to a change of the discovered process model because those deviations typically occur less frequently.

Algorithm 3: Adaptive sliding window algorithm for process drift detection.

input : An event log L that should be scanned
input : Significance level threshold
output: A list D of process drift positions in the event log

$w \leftarrow 100$; $maxSize \leftarrow 200$; $i \leftarrow 1$
while $i < |L| - w$ **do**
  $ref_{start} \leftarrow i$; $ref_{end} \leftarrow i + w - 1$
  $det_{start} \leftarrow i + w$; $det_{end} \leftarrow i + 2w - 1$
  if ***g-test***$(L[ref_{start} : ref_{end}], L[det_{start} : det_{end}]) <$ threshold **then**
    found $\leftarrow$ *false*; $w_{new} \leftarrow \frac{w}{2}$; lastI $\leftarrow det_{end}$
    for $j \leftarrow det_{start} - w_{new}$ **to** $det_{end} - w_{new} + 1$ **do**
      lastI $\leftarrow j + 2w_{new} - 1$
      $p_{val} \leftarrow$ **g-test**$(W[j : j + w_{new} - 1], W[j + w_{new} : j + 2w_{new} - 1])$
      if $\hat{p}_{val} - p_{val} < -0.5$ **then** break;
      $\hat{p}_{val} \leftarrow p_{val}$
      if $p_{val} <$ threshold **then** `// drift found?`
        $D \leftarrow D \cup \{j + w_{new}\}$
        $i \leftarrow j + w_{new}$
        found $\leftarrow$ *true*
      end
    end
    if ***not*** found **then** `// no drift found, reset window`
      $w \leftarrow w \cdot$ lastI $/ det_{end}$
      $i \leftarrow i + w$
    end
  else
    $w \leftarrow 1.2w$; `// increase window size`
  end
  if $w \geq maxSize$ **then**
    $w \leftarrow 100$; `// no drift found`
    $i \leftarrow i + maxSize$; `// reset and set new reference window`
  end
end

We compare and detect deviations between the discovered process models $P = (N, E)$ by calculating different graph metrics:

- *Number of activities $|N|$ and transitions $|E|$:* The number of activities and transitions are indicators for the change of executed activities. New activities lead to more, skipped activities lead to fewer nodes and edges.
- *Graph density:* The graph density is the ratio between activities and transitions. It serves as an indicator for a change to the process complexity.

New or deprecated process variants result in a higher or lower density value.

$$D = \frac{|E|}{|N| \cdot (|N| - 1)} \tag{5.1}$$

- *In- and out-degree:* With the in- and out-degree of each activity $|in(n_i)|$, $|out(n_i)|$ for $n_i \in N$, a change to specific activity sequences can be detected. This graph metric is relevant for characterizing the process drift.
- *Occurrence of activities and transitions:* The occurrence of activities and transitions, i. e., the number of cases that contain a specific activity or follow a specific transition, are indicators for gradual or incremental drifts. They measure the number of cases that follow other transitions through the process model due to the drift.

Graph metrics are calculated for both windows and stored in a vector, denoted as $\vec{r}$ for the reference window and $\vec{d}$ for the detection window. Each dimension of the vector corresponds to a single graph metric. For instance, the first dimension corresponds to the graph density. Note that the degree is calculated for each activity, and the occurrence is calculated for each activity and transition in the sub-log of the respective window.

5.3.3 Perform a G-Test on the Graph-Metrics

This section describes how statistical significance testing over the graph metrics is used to identify process drifts. In general, statistical significance testing is used in observational studies or experiments to assess evidence about some claim about the population from which the gathered sample data has been drawn. Every statistical significance test starts with a null hypothesis (H_0). In our case, the null hypothesis is that the two consecutive subsets of cases follow the same process behavior. That is, whether the observed distribution over the transitions in the detection window is equal to the expected distribution of transitions in the reference window. The null hypothesis is rejected if the probability value (p-value) is lower than a given significance level α. Whenever the null hypothesis is rejected, a potential process drift is reported.

We use the G-Test [McD09] in favor of the Chi-Squared test because the G-Test is more robust and recommended for smaller sample sets [McD09]. The p-value for the G-Test is calculated using the following formula:

$$G = 2 \sum_i O_i \cdot \ln\left(\frac{O_i}{E_i}\right) \tag{5.2}$$

where O_i is the observed occurrence of a transition i and E_i is the expected occurrence of a transition i under the null hypothesis. The sum is only taken over all non-zero occurrences.

In our conducted experiments, the occurrence of activity transitions as well as the graph metric for significance testing yielded the best results. However, all metrics are kept to characterize the process drift in a more detailed fashion (see Section 5.3.4).

5.3.4 Characterization of the Process Drifts

In this section, we describe how the graph metrics are used for characterizing detected process drifts, i. e., what kind of structural changes or modifications were made to the process. For instance, one might be interested in the activities that are executed before the drift but skipped afterward. The algorithm relies on the process model and the graph metrics that are already calculated in the previous step.

For characterizing the process drift, the in- and out-degree of activities are explored by the process drift detector. Changes to the in- and out-degree of activities indicate new or removed transitions between activities, and thus structural changes to the process execution. If there are differences between the reference and the detection window, the detector collects the number of cases following the changed transition. Collected information is used to annotate and highlight the process drift in the process model visually.

Example 5.1 illustrates the characterization of the process drifts on a simple example process.

Example 5.1

Consider an example process model which refers to a procurement handling process. First, a customer creates a *Purchase request* which is *Approved* before a *Purchase Order* is created. Next, the *Goods Receipt* is generated before the *Payment* of the ordered goods is performed. We add a process drift to the process by introducing a new transition from the activity *Purchase Order* to *Payment* such that the activity *Goods Receipt* can be skipped. Figure 5.3 shows a process model with the added process drift marked in red.

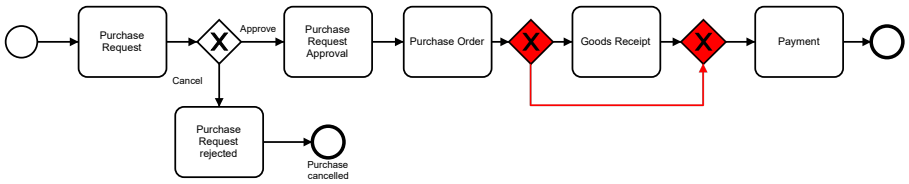


Figure 5.3: Process model in Business Process Model Notation (BPMN) with a process drift marked in red [SNM17].

We generated an event log that corresponds to the described procurement handling process, containing 2 000 cases and a process drift at case 1 000. Our process drift detection algorithm returns the following results:

Graph Metric	Change	Absolute Value
Number of transitions	-2	-
Number of activities	-1	-
In- and out-degree		
In-degree: Goods Receipt	-1	0
In-degree: Payment	-1	1
Out-degree: Purchase Order	-1	1
Out-degree: Goods Receipt	-1	0
Transition frequency		
Purchase Order → Goods Receipt	-16	0
Goods Receipt → Payment	-16	0
Purchase Order → Payment	14	24

Table 5.2: Graph metrics computed by the process drift detection algorithm.

Table 5.2 shows the detected changes to the graph metrics. The algorithm found that the *Goods Receipt* activity is skipped which results in a decrease of two transitions and a decrease of one activity. Consequently, the in- and out-degree of *Goods Receipt* is reduced by one, leading to a total value of zero. Note that the in-degree of *Payment* and the out-degree of *Purchase Order* is reduced by one, although only a single activity is removed. The reason is that the process drift detector did not precisely find the drift at case id 1 000, but at case id 1 020. As a result, 20 cases were put into the reference window, leading to a detection delay of 20 cases.

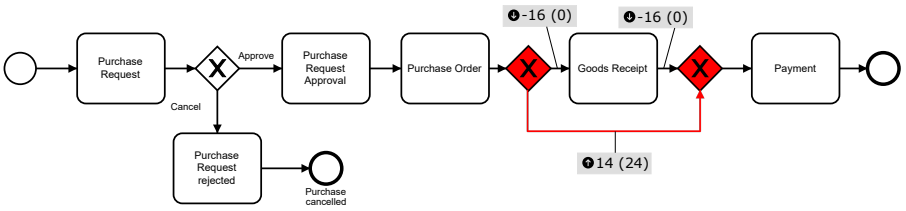


Figure 5.4: Resulting annotations to the process model in BPMN with a process drift marked in red.

The process drift detection algorithm uses the changes to the frequency of transitions between the reference and the detection window to reason about the actual process drift. According to Table 5.6, the number of cases that follow the transition between *Purchase Order* and *Goods Receipt* dropped from 16 to 0. Equally, the transition between *Goods Receipt* and *Payment* dropped from 16 to 0 cases. On the contrary, the number of cases that follow the transition between *Purchase Order* and *Payment* increased from 10 to 24 cases. The resulting process model with the annotations of the detected process drift is depicted in Figure 5.4.

5.4 Evaluation

This section presents various experiments that evaluate the performance of our process drift detection algorithm. The algorithm has been implemented as a ProM plugin [Ver+11] for evaluation purposes.

The evaluation of the process drift detector is twofold:

1. We compare the accuracy of the process drift detection algorithm with other related methods (see Section 5.4.2).
2. We compare the extracted characteristics with the injected process drifts of the event log to evaluate the validity of the provided results (see Section 5.4.3).

5.4.1 Experiment Setup

We use a benchmark data set of 72 synthetic event logs that was introduced in [Maa+15]. The benchmark is constructed from a base process model (see Figure 5.5) with 15 different activities. The base model is systematically modified to produce event logs with different patterns of process drifts (see Table 5.3). The modifications to the base model are categorized into three simple drift patterns: insertion ("I"), resequentialization ("R"), and optionalization ("O"). More complex process drift patterns are generated from the combination of simple drift patterns which result in additional event logs ("IOR", "IRO", "OIR", "ORI", "RIO", "ROI"). The generated event logs are composed of a fixed number of 9 process drifts, generated by alternating cases between the base and the modified process model.

In the evaluation, the event logs are varied in size (see Table 5.4). Each event log is annotated with the positions of the process drifts for evaluation purposes; these are not used for the detection.

For comparison, three other methods from the related work are applied to the same benchmark. First, the method of Bose et al. [Bos+14] is considered

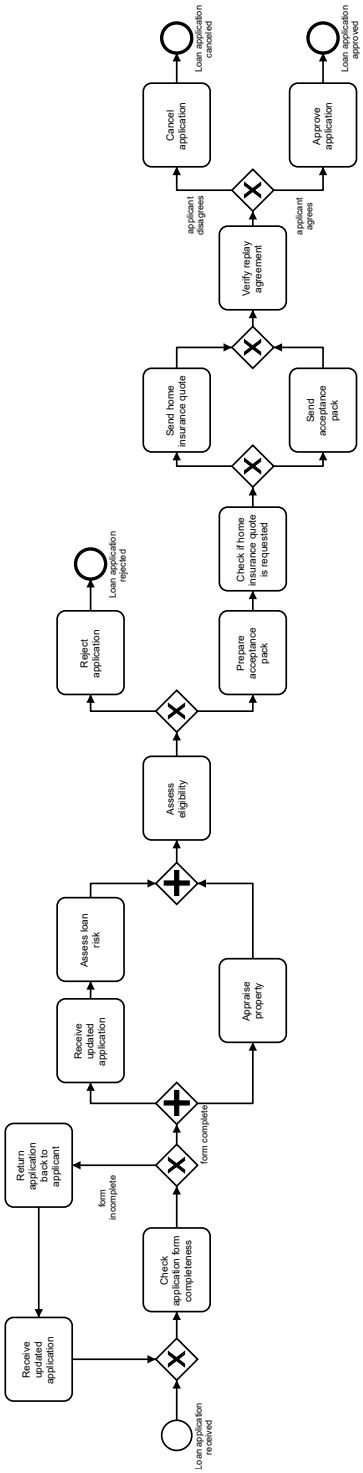


Figure 5.5: Base BPMN model of the benchmark dataset [Maa+15].

	Process Drift Pattern	Category
re	Add/remove fragment	I
cf	Make two fragments conditional/sequential	R
lp	Make fragment loopable/non-loopable	O
pl	Make two fragments parallel/sequential	R
cb	Make fragment skippable/non-skippable	O
cm	Move fragment into/out of conditional branch	I
cd	Synchronize two fragments	R
cp	Duplicate fragment	I
pm	Move fragment into/out of parallel branch	I
rp	Substitute fragment	I
sw	Swap two fragments	I
fr	Change branch frequency	O

Table 5.3: Process drift patterns of the benchmark event logs [WRR08; Maa+15].

	Bose et al.	Maaradji et al.	Ostovar et al.	Seeliger et al.
Log Size		$\lvert L\rvert = \{2500, 5000, 7500, 10\,000\}$		
Window Size	100	*auto*	100	*auto*
Noise Level	-	-	0.0	-
Sensitivity	-	-	very high	-

Table 5.4: Evaluation setup of all compared process drift detection methods.

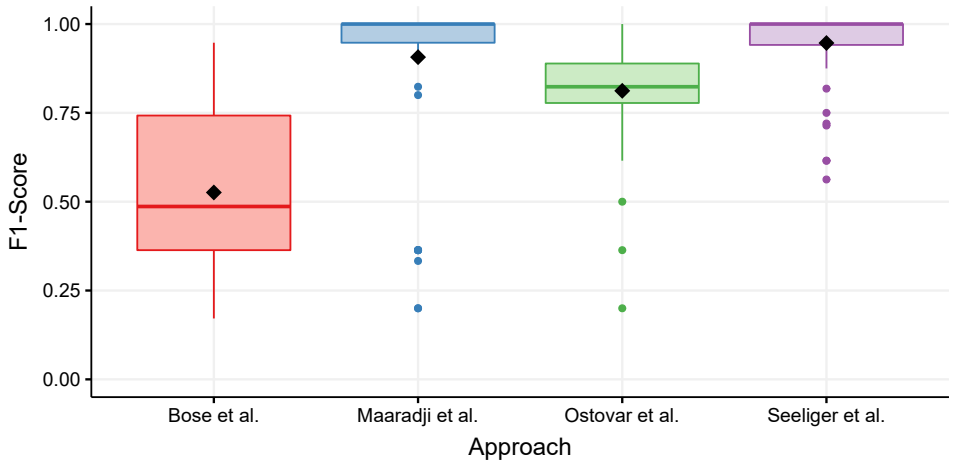


Figure 5.6: F1-score across all process drift patterns compared with the related work. Filled diamond (◆) indicates the mean.

which is based on a fixed sliding window. In the experiments, a fixed window size of 100 is used. Second, the adaptive sliding window method of Maaradji et al. [Maa+15] is used. Lastly, we compare the $\alpha+$ based method of Ostovar et al. [Ost+16] using the adaptive window starting with a size of 100, the noise level of 0.0, and a sensitivity of *very high*.

5.4.2 Accuracy Results

In the first part of the evaluation, we measure the F1-score and the average detection delay to evaluate the performance of the process drift detectors. In this context, precision is defined as the probability that the system detects a true process drift. The recall is defined as the probability that the detected process drift is a correct drift [Hoo5]. The average detection delay indicates how many cases early or late the detected drift deviates from the actual drift.

Figure 5.6 shows the F1-scores for the compared methods summarized across all process drift patterns. The detailed results are reported in Table 5.5. In the experiment, our algorithm gathered an overall F1-score of 0.9466 ($\sigma = 0.0971$), compared with the second best method from Maaradji et al. with 0.9068 ($\sigma = 0.2160$), followed by the method of Ostovar et al. with 0.8121 ($\sigma = 0.1393$), and the approach from Bose et al. 0.7011 ($\sigma = 0.3075$). For 16 out of 18 process drift patterns (see Figure B.1), our algorithm achieved F1-scores in the range between 0.9 and 1.0. Our process drift detection algorithm provides better overall performance results for the F1-score compared to the related work. Only for the process drift patterns *lp* and *OIR*, the related work produces better results than our algorithm. The reason is that the FHM does not detect the activity loop which results in no change to the process model, and thus to no detection of the process drift. Furthermore, the experiments

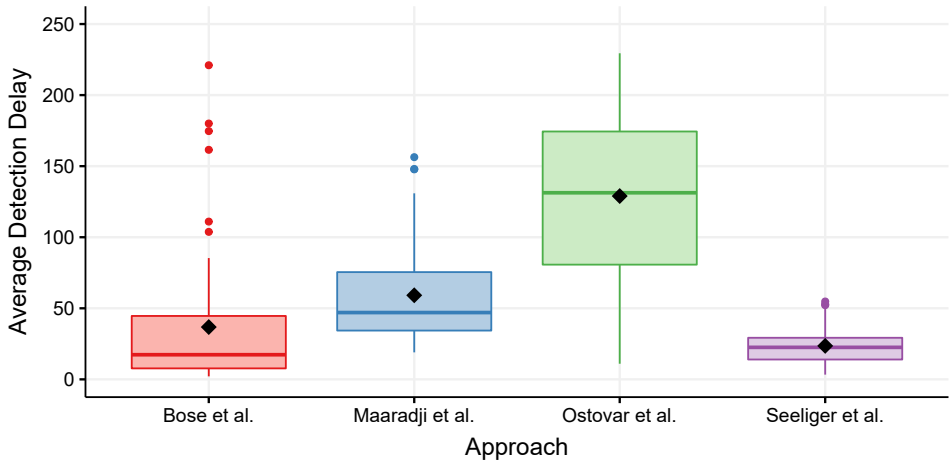


Figure 5.7: Average delay across all process drift patterns (lower is better). Filled diamond (◆) indicates the mean.

Approach	F1-Score		Delay	
	Mean	σ	Mean	σ
Bose et al.	0.7011	0.3075	41.17	37.56
Maaradji et al.	0.9068	0.2160	59.15	34.15
Ostovar et al.	0.8121	0.1393	234.95	212.37
Seeliger et al.	**0.9466**	**0.0971**	**23.56**	**12.92**

Table 5.5: Results of the experimental evaluation of the process drift detection methods.

revealed that the method of Bose et al. does not detect the process drift patterns *cb* and *cm*. The adaptive sliding window method of Maaradji et al. produces similar results as our algorithm, but it has issues with the process drift pattern *cd*.

In Figure 5.7 the average detection delays across all process drift patterns are depicted. The detailed results are reported in Table 5.5. Our process drift detection algorithm has the lowest detection delay with an average delay of 23.6 ($\sigma = 12.9$) cases. The method of Bose et al. is the second best with an average delay of 41.17 ($\sigma = 37.6$) cases. The third best is the method of Maaradji et al. with an average delay of 59.2 ($\sigma = 34.2$) cases, followed by Ostovar et al. with 235.0 ($\sigma = 212.4$). It is noteworthy that for the process drift pattern *fr* the average delay of the method of Bose et al. lies over 160 cases. This detection delay is quite high compared to the two other methods (see Figure B.2).

5.4.3 Characterization Extraction Results

In the second part of the evaluation, we evaluated the extracted characterizations of our process drift detection algorithm. For each of the process drift patterns (see Table 5.3), we compare the extracted characterizations with the real modifications of the benchmark dataset. Example 5.2 illustrates the comparison of the single process drift pattern *cb*.

Example 5.2

The *cb* process drift pattern introduces a new conditional transition from the activity *Assess eligibility* to the transition *Check if home insurance quote is requested*. This new transition should be found as the process drift characterization. Figure 5.8 shows the corresponding process model.

When running the algorithm, the following graph metric changes are observed:

Graph Metric	Change	Value
Out-degree: Assess eligibility	1	-
Out-degree: Check if home insurance qu...	-1	-
Assess eligibility → Prepare acceptance...	-8	16
Assess eligibility → Send home insurance...	10	10
Assess eligibility → Reject application	-7	26
Check if home... → Send home insurance...	-12	0
Check if home... → Send acceptance pa	-3	9

Table 5.6: Observed graph metric changes and absolute values of transitions for the given example.

The graph metric changes indicate that the activities *Assess eligibility* and *Check if home insurance quote is requested* are involved. The new transition from *Assess eligibility* to *Send home insurance quote* is recognized by the occurrence increase from 0 to 10 cases. At the same time, the transition *Check if home insurance quote is requested* to *Send home insurance quote* is removed, indicated by the change from 12 to 0. The new transition allows bypassing the activity *Prepare acceptance pack* and *Check if home insurance quote is requested*.

Table 5.7 shows the different characterizations extracted by our algorithm for each simple process drift pattern. A correctly identified characterization is marked with a checkmark (✓) whereas an incorrect characterization is marked with a cross mark (✗).

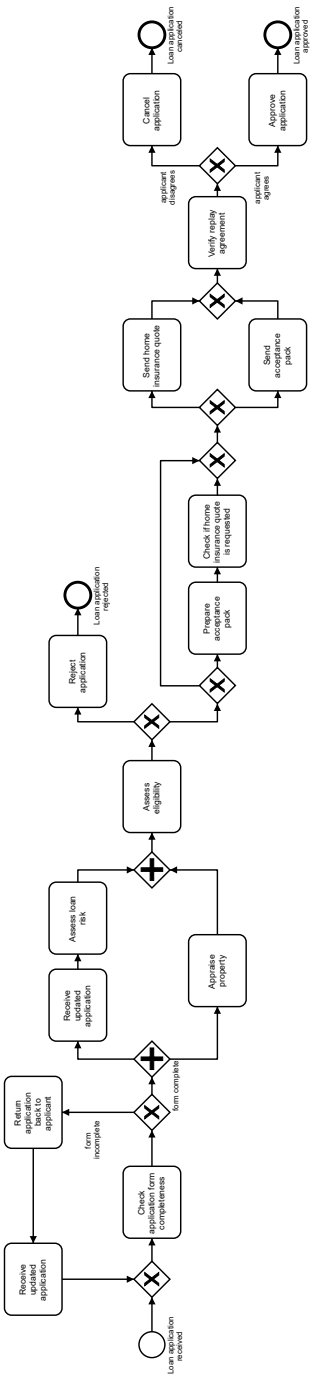


Figure 5.8: Process model of the *cb* process drift pattern event log [Maa+15].

	Characterization	Found
re	Remove of *Assess eligibility*	✓
cf	*Send acceptance pack* and *Send home insurance quote* are sequential	✓
lp	-	✗
pl	*Check credit history, Assess Loan risk* and *Appraise property* are sequential	✓
cb	New transition from *Check if home insurance quote is requested* to *Send home insurance quote*	✓
cm	*Check if home insurance quote is requested* followed by *Prepare acceptance pack*	✓
cd	New transition *Appraise property* to *Assess loan risk* Remove transition *Appraise property* to *Assess eligibility*	✓
cp	Detection of a loop	✗
pm	New transition *Check if home insurance quote is requested* to *Prepare acceptance pack* New transition *Prepare acceptance pack* to *Verify repayment agreement*	✓
rp	*Replace events*	✓
sw	Remove transition *Send acceptance pack* to *Verify repayment agreement* Remove transition *Assess eligibility* to *Prepare acceptance pack* Remove transition *Send home insurance quote* to *Verify repayment agreement* Add transition *Send home insurance quote* to *Prepare acceptance pack* Add transition *Send acceptance pack* to *Prepare acceptance pack* Add transition *Assess eligibility* to *Verify repayment agreement*	✓
fr	-	✗

Table 5.7: Characterizations extracted from the contributed algorithm for each process drift pattern.

The results show that our algorithm correctly identifies the characteristics of the process drift in 9 out of 12 process drift patterns. For three patterns the algorithm is not able to identify the correct characteristics; in two cases the algorithm is not able to identify any characteristics at all.

5.5 Discussion and Limitations

We presented a process drift detection algorithm based on process models to detect behavior changes. However, there are some limitations and research directions for future work.

5.5.1 Drift Types

Our process drift detection algorithm detects all four process drift types but cannot distinguish between them. However, it may be necessary to identify the drift type, such as recurring drifts, to reveal further insights. With the current hypothesis testing and adaptive sliding window approach, identifying the different process drift types is not possible.

Another limitation is the characterization of drifts, which only yields useful explanations for simple drift patterns. We found that process drifts containing fragment modifications, sequentialization, skipping, additional transitions or activities, and replacements were correctly characterized. However, complex drift patterns, such as multiple combinations of simple drift patterns or frequency changes, yield inaccurate and incomplete explanations. The use of graph metrics, as proposed in our approach to characterize complex drift patterns, does not seem sufficient to provide useful information here. In particular, we observed that in situations were a process drift leads to major structural changes, the results of our characterization approach are rather limited. It might be interesting to investigate context-aware process mining techniques to compare process models of different periods to enhance characterization.

Our approach is also limited to the control flow perspective to detect process drifts in event logs. Other perspectives, such as the organizational perspective, can also reveal process drifts that cannot be identified by analyzing the control flow only. A possible approach is to use the multi-perspective heuristics miner to also include other perspectives to the discovered process model [MDR17].

5.5.2 Process Model Discovery

The process drift detection algorithm discovers process models with the FHM algorithm. As a result, the quality of the detected process drift relies on the quality of the discovered models. In some situations, the FHM tends to

generate inaccurate and unsound process models which lead to inaccuracies of the calculated graph metrics. Highly complex process models with a low replay fitness can result in wrongly detected process drifts. In future work, a more accurate and reliable process discovery algorithm may be used to overcome the above issues.

5.5.3 Visualization

The current implementation only provides a basic visualization of the process drifts identified. A more sophisticated visualization method that shows how many instances have exactly followed the new path through the process model may help to better understand the process drift. Only the instances of the consecutive windows are currently used to calculate the visual process model annotations but these only represent a small subset of cases.

5.6 Conclusion

In this chapter, we presented a novel process drift detection algorithm that enables the detection and characterization of process drifts from event logs. In summary, the main contributions of this chapter are:

1. A parameter-free process drift detection algorithm that is based on a custom-designed adaptive sliding window approach. It evaluates the deviation of discovered process models by performing statistical significance tests on certain graph metrics.
2. A method to extract process drift explanations using graph metrics of discovered process models. It identifies which activities and transitions have changed, and how many cases follow the identified new behavior.
3. Results from an experimental evaluation investigating the detection accuracy and delay of the algorithm introduced in this chapter.

The process drift detection algorithm introduced in this chapter identifies changes to the process behavior over time and provides insights about what has changed. As illustrated in Figure 5.9, process drift detection is based on the process knowledge artifact framework and autonomously obtains valuable insights about changes to the process. Identified process changes help analysts better understand the process by pointing to interesting time periods when the process has changed. Process behavior can not only change over time, but different process behavior may also occur simultaneously. This is the case when different departments execute the process differently. However, this different behavior cannot be identified by process drift detection. In the following chapter, we introduce a multi-perspective trace clustering algorithm that can split an event log into subsets of cases with similar behavior to close this gap.

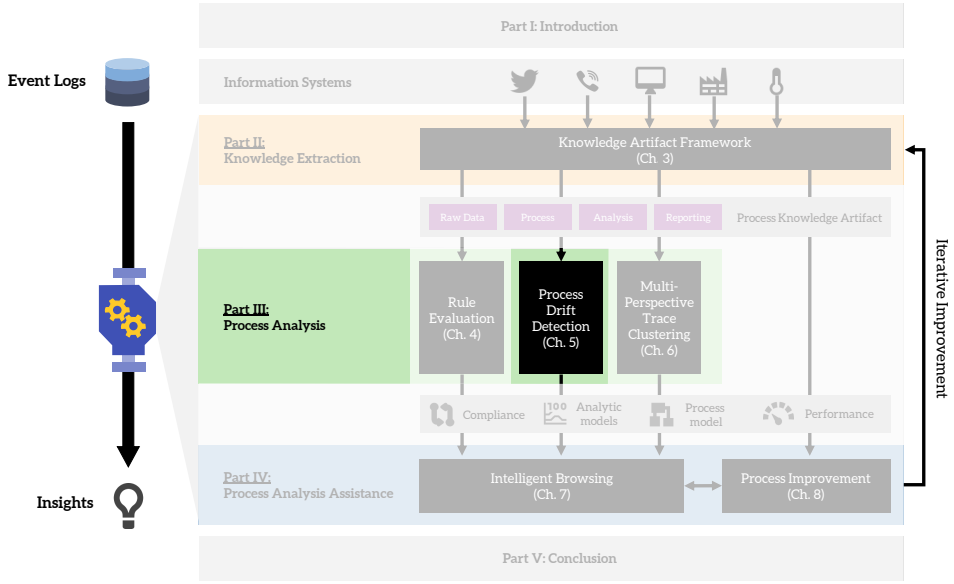


Figure 5.9: Overview of the chapter and contributions.

6 Hybrid Feature Set Trace Clustering

The previous chapter presented a process drift detection algorithm, identifying changes in process behavior over time. This chapter introduces a trace clustering algorithm that splits an event log into subsets of cases with similar process behavior that occur simultaneously, i. e., at the same time. In particular, we introduce *Hybrid Feature Set Trace Clustering*, which is a multi-perspective trace clustering algorithm that combines the control flow and the data perspectives. On the one hand, the edit distance between activity sequences is used to calculate the similarity on the control flow perspective. On the other hand, the algorithm uses frequent itemset mining to identify co-occurring case attributes to find similarities of cases on the data perspective. Our algorithm maximizes the fitness of the corresponding subset process models to find the optimal balance between both similarity metrics. In particular, it automatically optimizes the number of clusters, the balance between the considered process perspectives, and the parameter for the frequent itemset mining.

This chapter is organized as follows: First, a short introduction to trace clustering is given in Section 6.1. Next, the related work in the context of trace clustering is discussed (Section 6.2). Section 6.3 describes our trace clustering algorithm that incorporates the control flow as well as the data perspective to generate clusters. In Section 6.4 the evaluation of the algorithm is presented. The chapter concludes with the limitations of the presented approach and potential future work (Section 6.5).

> **Publication:** This chapter is based on the following publications:
>
> Alexander Seeliger, Benedikt Schmidt, Immanuel Schweizer, and Max Mühlhäuser. "What Belongs Together Comes Together. Activity-centric Document Clustering for Information Work." In: *Proceedings of the 21st International Conference on Intelligent User Interfaces - IUI '16*. New York, NY, USA: ACM Press, 2016, pp. 60–70. DOI: 10.1145/2856767.2856777.
>
> Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. "Finding Structure in the Unstructured: Hybrid Feature Set Clustering for Process Discovery." In: *Business Process Management*. Cham: Springer International Publishing, 2018, pp. 288–304. DOI: 10.1007/978-3-319-98648-7_17.

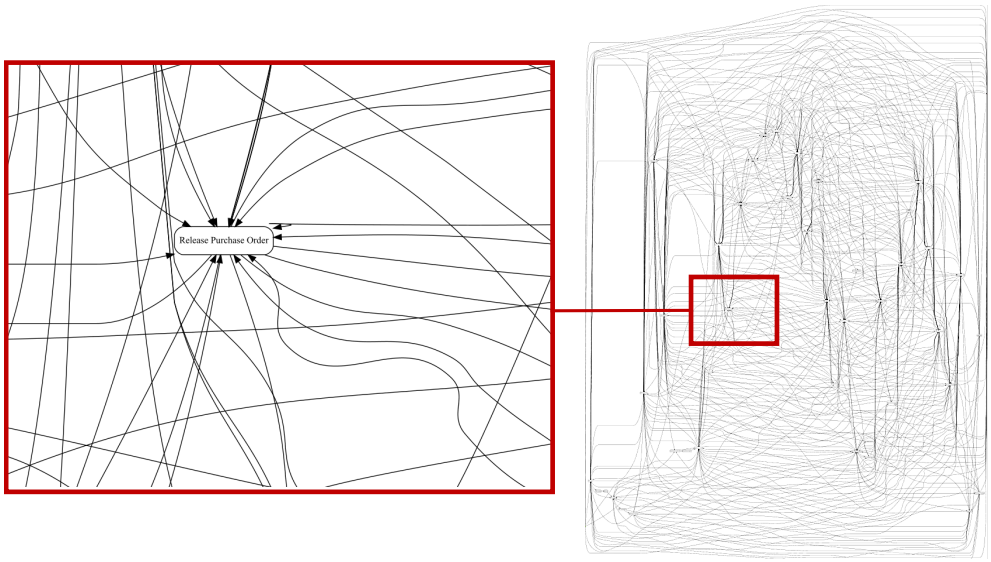


Figure 6.1: Process map of the BPI Challenge 2019 event log showing all observed activities and transitions.

> **Contribution Statement:** I led the idea generation, implemented the prototype and performed the data evaluation. *Timo Nolle*, *Benedikt Schmidt*, *Immanuel Schweizer* and *Max Mühlhäuser* supported the conceptual design and contributed to the writing process.

6.1 Introduction and Motivation

Process discovery is an essential part of process mining because it helps analysts and organizations to understand the actual use of their supporting Process-aware Information System (PAIS) [Aal11]. Many process discovery algorithms (see Section 2.4) are designed for relatively structured processes, but many of today's business processes are executed in flexible environments [BA09; VF10; Goe+11; RW12]. For instance, processes in health care, product development, or customer service have a high density of cases with a high variety of complex behavior. Consequently, two challenges need to be addressed:

1. The high variability of process executions leads to highly complex and inaccurate process models [BA09; VF10; Aal16] (see Figure 6.1). Such discovered process models are hard to understand for analysts [MRC07], and navigating through them requires extensive knowledge about the process. That is why they are also often called spaghetti models.
2. Behavioral differences on other process perspectives (e. g., case attributes; see Section 2.2.3) are neglected, i. e., process discovery mostly only considers the control flow perspective of a process to obtain pro-

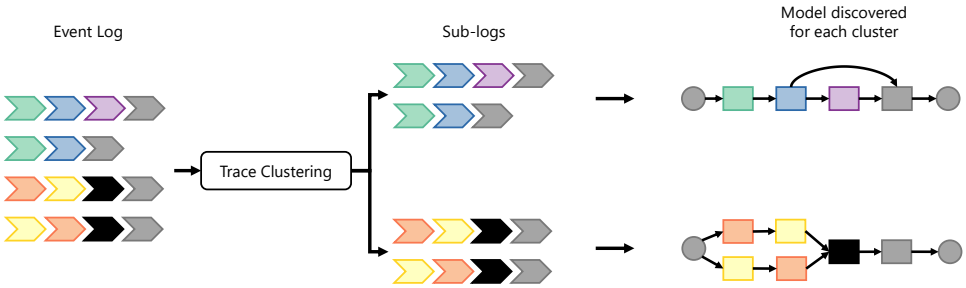


Figure 6.2: Overview of trace clustering applied to a single event log.

cess models. For instance, patients in a hospital go through the same admission process, but they are then divided into emergency and non-emergency patients. Although the executed sequence of activities is similar, patients are treated by different staff which may influence throughput time. As a result, process models discovered by algorithms that only consider the control flow may show misleading duration times.

Trace clustering aims to address both issues by splitting the event log into a set of similar sub-logs, i. e., a subset of cases with similar behavior (see Figure 6.2). Instead of applying process discovery to the entire event log, it is applied to each of the sub-logs individually, resulting in a set of process models that reflect the different process behaviors. It has been shown that these process models can have better fitness and precision [SGA09; Wee+13; SB16], and they can be less complex because less process behavior is contained [SB15]. However, existing trace clustering methods either require additional information, e. g., the number of clusters to be generated, do not optimize their clustering concerning process model quality, i. e., optimizing for model fitness, or neglect other process perspectives, e. g., differences in process behavior on the data perspective.

This chapter introduces a multi-perspective trace clustering algorithm that automatically optimizes the resulting process models for fitness, incorporating both control flow and data perspectives. The decision to consider these two perspectives follows the observation that the behavior of a case typically depends on the context it is executed. We define two separate metrics, one for the control flow and one for the data perspective, that are combined by a weighting factor. The weighting factor is obtained automatically by solving an optimization problem, that maximizes the fitness of the resulting process models. As a result, process models discovered by our multi-perspective trace clustering algorithm accurately reflect the different process behaviors that exist in the event log.

6.2 Related Work

This section introduces related work in the context of trace clustering. The related work of trace clustering in process mining is categorized into (1) similarity- and (2) model-based clustering approaches.

We compare the related work based on a set of relevant requirements for designing trace clustering algorithms:

(R1) **Optimize process model accuracy**
The clustering result of trace clustering in process mining should be guided by optimizing the accuracy of the resulting process models. It is one of the four quality metrics in process mining to evaluate process discovery [BDA14] (see Section 2.4.3).

(R2) **Support multiple process perspectives**
The control flow of activities is the basis for trace clustering in process mining, but other perspectives should also be considered. Processes should be seen as multi-perspective entities in which process behavior is also only encoded into the case attributes of an event log [Jab+19].

(R3) **Provide cluster explanations**
The trace clustering algorithm should provide human understandable explanations on why specific clusters are generated [DDB16]. Without any explanations, clustering results may be difficult to interpret because their discrimination is not visible to the analyst.

(R4) **Be parameter-free**
The trace clustering algorithm should be parameter-free, i. e., it should only use an event log as the input. Clustering usually requires the setting of appropriate parameter values, e. g., the number of clusters to be generated. This is typically not possible because there is no knowledge about the different process behaviors that are present in the event log.

Table 6.1 compares the relevant research in each group regarding the established requirements. A different comparison of similar related work is introduced by Thaler et al. [Tha+15], who show some of the same observations as presented in this section. The following sub-sections elaborate on the related work in detail.

6.2.1 Similarity-based Trace Clustering

The fundamental idea of similarity-based trace clustering is to project each case of an event log into the vector space and determine the similarity between two cases using a similarity function. The underlying assumption is that similar cases are also close together in the vector space. An early

	Optimize process model accuracy	Support multiple process perspectives	Provide cluster explanations	Be parameter-free
Similarity-based Trace Clustering				
Greco et al. [Gre+06]	○	○	○	○
Bose and van der Aalst [BA09]	○	○	○	○
Bose and van der Aalst [BA10]	○	○	○	○
Song et al. [SGA09]	○	●	○	○
Hompes and Buijs [HB15]	○	●	○	●
Evermann et al. [ETF16]	●	○	○	○
Appice and Malerba [AM16]	○	●	○	○
De Koninck et al. [DDB16]	○	○	●	○
Yang et al. [Yan+16]	○	●	●	○
De Koninck et al. [De +17]	○	○	○	○
Delias et al. [Del+17]	○	●	○	○
De Koninck and De Weerdt [DD17]	◐	◐	○	●
Fani Sani et al. [San+17]	○	●	●	○
Wang et al. [Wan+18]	○	○	○	○
Jablonski et al. [Jab+19]	○	●	○	○
Model-based Trace Clustering				
Ferreira [Fer09]	○	○	○	○
Veiga and Ferreira [VF10]	○	○	○	○
De Weerdt et al. [Wee+13]	●	○	○	○
Ekanayake et al. [Eka+13]	●	○	○	○
Sun et al. [SBW17]	●	○	○	○
Chatain et al. [CCD17]	○	○	○	○

Table 6.1: Property comparison of trace clustering algorithms. ● indicates the fulfillment, ◐ indicates a partial fulfillment, and ○ indicates the non-fulfillment of a requirement.

approach to discovering expressive process models through clustering was presented by Greco et al. [Gre+06]. Here, traces sharing similar subsequences of activities are clustered together using the k-means clustering algorithm (see Section 2.5.1). Different sequence feature vector encodings based on the edit distance are explored and compared by Bose et al. [BA09; BA10]. Due to the high sensitivity of the edit distance, the authors propose an automated algorithm for deriving the edit costs.

Another research direction uses sequence alignment-based methods to calculate the similarity between traces. Evermann et al. [ETF16] apply the Smith-Waterman-Gotoh algorithm for sequence alignment to compute the similarity between traces. After the similarity scores are adjusted by applying multi-dimensional scaling, k-means clustering is used to obtain similar cases. A further improvement to the clustering approach is introduced by Wang et al. [Wan+18], who focus only on relevant activities. In that paper, the authors introduce a constraint trace similarity metric based on activity dependencies. In [DD17] the authors explore the stability of the clusters to determine the appropriate number of clusters by iteratively comparing the similarity between cluster results of perturbed and unperturbed event logs. The number of clusters is then determined by the most stable results.

Multiple process perspectives are considered by Song et al. [SGA09] who introduce trace profiles. A trace profile is defined as a feature vector that encodes different features and perspectives of a trace. For instance, duration times between activities, resource allocation, and the order of activities are separate trace profiles which can be individually selected for clustering. The same authors evaluated different dimensionality reduction methods for feature selection to improve clustering performance [Son+13]. In [HB15] the authors discover deviating cases and process variants using the Markov cluster algorithm. A co-training strategy that combines multiple trace profiles is proposed by Appice et al. [AM16]. The clustering result of a single trace profile is constrained by the similarities of all other considered trace profiles which iteratively converges towards a unique clustering pattern. VITPLA [Yan+16] summarizes traces on multiple perspectives by aligning traces and finding associations between trace clusters and trace attributes. Jablonski et al. [Jab+19] introduce a multi-perspective trace clustering approach with the ability to control the contribution of the different perspectives. However, selecting appropriate weighting parameter values remains the task of the user.

Considering different heterogeneous scaled similarity criteria can become an issue when multiple criteria are combined. Delias et al. [Del+17] propose an outranking approach that overcomes this issue by introducing an overall combined similarity metric. Each criterion is assigned a weight that reflects the voting power in the overall similarity metric. The outranking method

ensures that individual criteria that do not match the majority of the other criteria are ignored.

De Koninck et al. [De +17] incorporate expert knowledge to improve clustering results. The basic idea is to guide the algorithm towards a clustering result that is much more consistent with the user's expectations. This can be achieved by initializing the clustering with a partially clustered dataset, either constraining the clustering or combining the clustering result derived with expert knowledge with the regular clustering solution. A similar expert-driven approach is proposed by Fani Sani et al. [San+17]. The approach uses a subgroup discovery algorithm that is guided by previously selected target variables.

A major issue with similarity-based trace clustering approaches is that they do not consider the quality of the process models to generate clusters. Consequently, clustering results are not optimized for process mining purposes, for instance, process discovery. Most similarity-based approaches also lack the ability to provide cluster explanations, or they need the analyst to set appropriate parameter values to obtain valuable results.

6.2.2 Model-based Trace Clustering

Model-based trace clustering incorporates the quality of the resulting process models, i. e., the process models discovered from the sub-logs, to guide the cluster generation. One of the first model-based trace clustering approaches is ActiTraC [Wee+13]. ActiTraC optimizes the result of the clustering such that the fitness (see Section 2.4.3.1) of each of the discovered process models is maximized. Instead of defining a similarity function between traces, candidate traces are iteratively assigned to a cluster. The quality of the cluster is determined by applying the Flexible Heuristics Miner (FHM) (see Section 2.4.2) and evaluating the fitness. The work presented in [SB15] extends this work by adding the process model complexity of the resulting clusters as a quality measure. A similar approach is presented by Sun et al. [SBW17] who address the stability issue of ActiTraC. The average complexity of the models is optimized before process models are optimized for fitness [SB16].

A first-order Markov Chain trace clustering method which models the sequences of activities is introduced in [Fero9; VF10]. For each of the clusters, a separate Markov Chain is generated to determine the probability that a trace is generated by this cluster. Traces are assigned to the clusters with the highest probability. Chatain et al. [CCD17] introduce a trace alignment clustering that is guided by the alignment with an existing process model.

Ekanayake et al. extend the idea of trace clustering by additionally slicing the resulting process models into sub-process models to further improve the comprehension of the models [Eka+13].

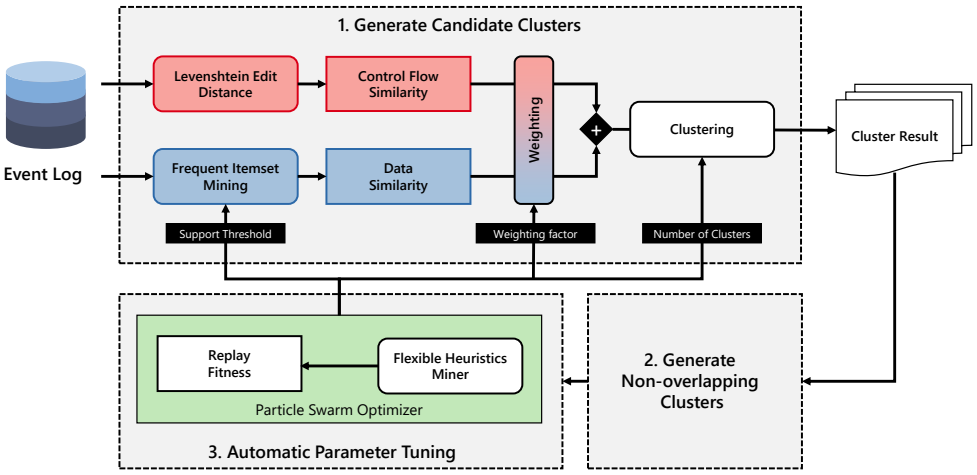


Figure 6.3: Architectural overview of the hybrid feature set trace clustering algorithm.

Model-based trace clustering approaches tend to provide better results from a process mining perspective, i. e., they optimize the accuracy of the underlying process models of each cluster. However, incorporating other perspectives than the control flow is not possible. Additionally, generated clusters are typically not explainable in a human-friendly fashion, and, similar to similarity-based approaches, they need the analyst to set appropriate parameter values.

6.3 Hybrid Feature Set Clustering

In this section, we introduce a multi-perspective trace clustering algorithm that combines the similarity of cases on the control flow as well as on the data perspective. The iterative algorithm is divided into three steps:

1. Candidate clusters are generated using a combined similarity function for the given event log (Section 6.3.1).
2. The clustering result of the previous step is used to create non-overlapping clusters by removing duplicate cases from the clusters with the least similarity (Section 6.3.2).
3. A novel optimization algorithm is applied, maximizing the fitness of each of the mined process models to determine the optimal balance between control flow and data perspective (Section 6.3.3).

Figure 6.3 shows the overall architecture and the performed steps of the hybrid feature set trace clustering algorithm.

In the following, each step is described in detail.

6.3.1 Generation of Candidate Clusters

In the first step, we define similarity functions that determine the similarity between cases from the event log. We introduce two different similarity functions, one for the control flow and one for the data perspective, i. e., the case attributes. The two similarity functions are then combined for the generation of candidate clusters.

Definition 6.1 (Similarity Function)**.** A similarity function on a set of cases $\mathcal{C}$ is a function

$$sim : \mathcal{C} \times \mathcal{C} \to [0, 1]$$

where $[0, 1]$ is the set of non-negative real numbers between 0 and 1, and for all $x, y \in \mathcal{C}$, the following conditions hold:

1. $0 \leq sim(x, y) \leq 1$
2. $sim(x, y) = 1 \Leftrightarrow x = y$ (identity of indiscernibles)
3. $sim(x, y) = sim(y, x)$ (symmetry)

The similarity function returns values near 0 if the two cases are very dissimilar, and values near 1 indicate very similar cases.

We construct a hybrid similarity function according to Definition 6.1 by combining two separate similarity functions as follows:

- The *control flow similarity function* sim_{lev} calculates the similarity between cases by inspecting the sequence of activities (see Section 6.3.1.1).
- The *case attribute similarity function* $sim_{itemsets}$ determines the similarity between cases by investigating the case attribute relationships (see Section 6.3.1.2).

The hybrid similarity function joins both separate similarity functions by a weighting factor (see Section 6.3.1.3).

6.3.1.1 Control Flow Similarity Function

We obtain the similarity between traces using the *Levenshtein edit distance* [Lev66]. The Levenshtein edit distance between two traces is defined as the minimum costs associated with transforming a trace to another one by performing specific edit operations. The edit operations are insertion, deletion, and substitution of an activity in the trace, each with the cost of 1. We use the Levenshtein edit distance because we want to obtain process models that have a high fitness. The edit distance has the characteristic that small differences between traces have only a small effect on the distance. At the same time, it is still sensitive to permutations, i. e., activities are performed in a certain order, and loops, i. e., the repetition of activities.

During the clustering, very similar traces may still be separated because the edit distance at least gives them a small distance. This allows the process discovery algorithm to generate process models with a higher fitness and lower complexity.

Definition 6.2 (Levenshtein edit distance)**.** Let L be the event log, $\hat{L}$ be the set of traces of L, and $X^i, Y^j \in \hat{L}$ be two traces of length i and j. We denote X^{i-1} as the subsequence of X^i with the first $i-1$ activities. We define the Levenshtein edit distance as follows:

$$lev(X^i, Y^j) = \begin{cases} \max(i,j) & if\ \min(i,j) = 0 \\ \min \begin{cases} lev(X^{i-1}, Y^{j-1}) + s(X(i), Y(j)) \\ lev(X^{i-1}, Y^j) + 1 \\ lev(X^i, Y^{j-1}) + 1 \end{cases} & otherwise \end{cases} \tag{6.1}$$

where $s(X(i), Y(i))$ is an indicator function that equals to 0 if $X(i) = Y(j)$, and otherwise to 1. The three calculations of the inner minimum function refer to the substitution, deletion, and insertion of activities. The outer maximum function calculates the remaining costs if the sequences are of unequal length.

We denote $sim_{lev}(X^i, Y^j)$ as the normalized Levenshtein distance which divides the costs by the maximum length of the traces. Example 6.1 illustrates the Levenshtein edit distance between two traces using Equation 6.1.

Example 6.1

Let $t_1 = \langle a, b, c, d, e \rangle$ and $t_2 = \langle a, b, g, d \rangle$ be two traces. The Levenshtein distance between t_1 and t_2 is $lev(t_1, t_2) = 2$.

		a	b	c	d	e
	0	1	2	3	4	5
a	1	0	1	2	3	4
b	2	1	0	1	2	3
g	3	2	1	1	2	3
d	4	3	2	2	1	2

Figure 6.4: The Levenshtein distance matrix which shows the calculation of the costs and the minimum cost path.

The corresponding matrix that shows the calculation is depicted in Figure 6.4. A diagonal jump in the matrix corresponds to the substitution operation (cf. first line of Levenshtein edit distance equation 6.1).

> Moving from one cell to another horizontal or vertically correspond to the deletion or insertion operation.

6.3.1.2 Case Attribute Similarity Function

We determine the similarity of case attributes by investigating attribute-value dependencies. The vector space that spans all attribute-value pairs of an event log can become huge because many unique combinations of case attributes and values can occur. Instead of comparing the case attributes individually, the relationships between case attributes are explored. This approach has two advantages:

1. The vector space of attribute-value pairs is much smaller.
2. Hidden attribute-value relationships between cases are revealed, which can be used to provide cluster explanations.

The algorithm obtains the reduced vector space by mining frequent itemsets from the case attributes that share the same sequence of activities. In particular, the *FPclose algorithm* (see Section 2.5.2.2) is used to mine closed frequent itemsets to limit the number of itemsets that are extracted from the event log given a certain *minimum support threshold* θ.

Each attribute-value pair of a case is converted into an index using integer encoding $\mathcal{I}_a : \mathcal{V}_a \to \mathbb{N}$. The integer encoding function assigns each attribute-value pair a unique index. For converting all attribute-value pairs of a case $c \in L$, we introduce a helper function $encode(c)$ which applies $\mathcal{I}_a$ to all attribute-value pairs of a case:

$$encode(c) = \{\mathcal{I}_a(\#_a(c)) : a \in \mathcal{A}\} \tag{6.2}$$

$encode(c)$ returns a set of indices which correspond to the attribute-value pairs of case c.

All cases following the same sequence of activity, i. e., process variants, are encoded and used as input for the FPclose algorithm. As a result, we obtain sets of itemsets, i. e., indices, that frequently occur together. This approach reveals certain case attribute patterns among cases of a process variant.

Let $S \subseteq \mathcal{S}$, $\mathcal{S}$ being the set of all possible itemsets, and $itemsets : \hat{L} \to \mathcal{P}(S)$ be the function that applies FPclose and returns sets of itemsets for a given trace $t \in \hat{L}$.

We determine the similarity of two sets of itemsets $S_a, S_b \subseteq S$ by the function $sim_{itemsets}$ that returns the relative amount of itemsets that are contained in both sets.

$$sim_{itemsets}(S_a, S_b) = \frac{2 \cdot |S_a \cap S_b|}{|S_a| + |S_b|} \tag{6.3}$$

6.3.1.3 Combined Similarity Function for Clustering

We define a combined similarity function that merges the output of the two similarity functions to obtain the similarity matrix for clustering. However, the two similarity functions deal with different types of input. The control flow similarity function sim_{lev} consumes traces and the case attribute similarity function $sim_{itemsets}$ consumes itemsets derived from the frequent itemset miner. We introduce a helper function that returns traces from a given itemset, defined as follows:

$$traces(s) = \{t : t \in \hat{L} \wedge s \in itemsets(t)\} \tag{6.4}$$

The combined similarity function $sim(s_a, s_b)$ is constructed as follows:

$$\begin{aligned} sim(s_a, s_b) = w \cdot sim^*_{lev}(traces(s_a), traces(s_b)) \\ + (1 - w) \cdot sim_{itemsets}(s_a, s_b) \end{aligned} \tag{6.5}$$

where $s_a, s_b \in S$. The weighting factor w of the combined similarity function controls the balance between both similarity functions. It is noteworthy that even if $w = 1$, the similarity function still incorporates the itemset similarity because the combined similarity function is defined over itemsets. As a result, cases that have the same set of itemsets will be clustered together.

Algorithm 4 shows the entire multi-perspective trace clustering specification including the definition of the similarity functions, the combined similarity function, and the clustering. First, all distinct traces $\hat{L}$ of the event log L are extracted by only considering the sequence of activities and applying the activity classifier (see Definition 2.4) to each event (Line 1). Next, Lines 2 and 3 define the similarity function for the control flow (see Section 6.3.1.1). sim^*_{lev} defines the similarity between two sets of cases, calculating the pair-wise similarity between traces and returns a normalized similarity value. Lines 4 - 8 refer to the encoding of the case attributes as well as the similarity function for the data perspective (see Section 6.3.1.2). The combined similarity function (see Section 6.3.1.3) is defined in Line 10 which calculates the similarity matrix over all pairs of itemsets.

Finally, the similarity matrix M and the number of clusters n are the input for the actual clustering algorithm. We use the *Agglomerative Hierarchical Clustering* algorithm (see Section 2.5.1.2) with ward-linkage to obtain the candidate clusters. Line 13 defines the set of clusters that contain the corresponding cases.

6.3.2 Generation of Non-Overlapping Clusters

The construction of Algorithm 4 leads to overlapping clusters because clusters are based on itemsets and not on cases. We address this problem by removing duplicate cases from the clusters where the similarity to all other cases is

Algorithm 4: Algorithm to retrieve the clusters [SSM18].

1 Let L be the event log, and let $\hat{L} = \{\underline{(\hat{c})} : c \in L\}$ be the set of distinct event traces of L.

2 Define $lev(x, y)$ as the edit distance of the event traces $x, y \in \hat{L}$.

3 Define $sim^*_{lev}(X, Y)$ as the edit distance between two sets of traces $X, Y \subseteq \hat{L}$:

$$sim^*_{lev}(X, Y) = \sum_{x \in X} \sum_{y \in Y} lev(x, y) \ / \ (|X| \cdot |Y|)$$

4 Define $cases(t) = \{c : c \in L \wedge \underline{(\hat{c})} = t\}$ as the cases following trace $t \in \hat{L}$.

5 Define $encode(c) = \{\mathcal{I}_a(\#_a(c)) : a \in \mathcal{A}\}$ with $c \in L$ and $\mathcal{I}$ as an integer index function; further let $encodes(C) = \{encode(c) : c \in C\}$.

6 Let $\mathcal{S}$ be the set of all possible itemsets, $S \subseteq \mathcal{S}$ and s_i be the i-th itemset in S.

7 Define $itemsets : \hat{L} \to \mathcal{P}(S)$ as the function that returns the frequent itemsets using the *FPclose* algorithm with θ being the *minimum support threshold*:

$$itemsets(t) = FPclose(encodes(cases(t)), \theta)$$

8 Define $sim_{itemsets}(S_a, S_b)$ as the similarity function of the itemsets with $S_a, S_b \subseteq S$:

$$sim_{itemsets}(S_a, S_b) = \frac{2 \cdot |S_a \cap S_b|}{|S_a| + |S_b|}$$

9 Define $traces(s) = \{t : t \in \hat{L} \wedge s \in itemsets(t)\}$ with $s \in S$ to be the inverse function of *itemsets* which returns the traces for a given itemset.

10 Define $sim(s_a, s_b)$ as the combined similarity function with $s_a, s_b \in S$, $w \in \mathbb{R}$ and $0 \leq w \leq 1$ to be the weighting factor:

$$sim(s_a, s_b) = w \cdot sim^*_{lev}(traces(s_a), traces(s_b)) + (1 - w) \cdot sim_{itemsets}(s_a, s_b)$$

11 Define $M : S \times S \to \mathbb{R}^{|S| \times |S|}$ as the itemset similarity matrix

$$M = (m_{ij}) = sim(s_i, s_j)$$

12 Let $cluster(M, n)$ be the hierarchical clustering function that returns the cluster index of each itemset as a vector of size $|S|$ with $n \in \mathbb{N}$ being the number of clusters to generate.

13 Define $C(k)$ to be the set of the traces in cluster k

$$C(k) = \{cases(traces(s_j)) \mid s_j \in S \wedge c_j \in cluster(M) \wedge c_j = k\}$$

lowest regarding the control flow. Algorithm 5 shows how the overlaps are resolved. The algorithm is repeated as long as there are no more cases to remove.

Algorithm 5: Algorithm to resolve overlapping clusters

```
for C(i) ∈ C do
    for l̂ ∈ C(k) do
        for C(j) ∈ C do
            if C(i) = C(j) then
                continue
            end
            if l̂ ∈ C(j) ∧ sim_lev(C(i), l̂) < sim_lev(C(j), l̂) then
                C(j) ← C(j) \ l̂
            end
        end
    end
end
```

As a result, cases that were previously assigned to multiple clusters are only contained in the cluster with the most similar control flow. This heuristic contributes to the goal of obtaining accurate process models for each cluster, since common process discovery algorithms reconstruct process models only based on the control flow.

6.3.3 Automatic Parameter Tuning

In the third and final step, the problem of finding appropriate parameters for the algorithm is addressed. Most related trace clustering methods do not provide any solution to this issue and solely rely on the expertise of the user to make the decisions, i. e., trace cluster algorithms are not parameter-free (see Table 6.1). However, the selection of appropriate parameter values heavily influences the clustering result. Parameter values that work well for one event log may not necessarily lead to good results for other event logs. Therefore, parameter values need to be set for each event log individually. We design an automatic parameter tuning approach to address this issue. It automatically determines the best weighting factor of the combined similarity function w, the minimum support threshold θ, and the number of clusters n. As a result, our multi-perspective trace clustering algorithm only requires an event log to perform trace clustering.

We construct an optimization problem to automatically obtain appropriate parameter values. The optimization problem considers the following three criteria:

- *Fitness.* The fitness measure (see Section 2.4.3.1) reflects the ability of a process model to reproduce the process behavior contained in the event

log. Maximizing the fitness leads to accurate process models that better reflect the observed behavior recorded in the event log.

Definition 6.3 describes how the weighted fitness is calculated.

Definition 6.3 (Weighted Fitness)**.** Let $fitness_k$ the fitness of a model k and n_k the number of cases in k, then the weighted fitness is calculated as:

$$\text{Weighted Fitness} = \frac{\sum_{k=1}^{N}(n_k \cdot fitness_k)}{|L|} \tag{6.6}$$

- *Silhouette Coefficient.* The silhouette coefficient (see Section 2.5.1.3) measures how well clusters are separated from each other. Maximizing the silhouette coefficient value leads to clusters with cases that have a high similarity to other cases in the cluster and a low similarity to objects in other clusters.
- *Number of Clusters.* Without an additional boundary condition, the optimizer algorithm increases the number of clusters to maximize the fitness of the models. In particular, assigning each trace to a separate cluster results in a fitness of 1 because each trace can be correctly replayed by its corresponding process model. Such behavior is not a desired property of the entire trace clustering algorithm.

The process model for obtaining the fitness is constructed by the FHM [WR11]. We choose the FHM algorithm because it is computationally efficient and robust against noisy event logs. However, it should be noted that the FHM may not always provide a good abstraction of the given event log, leading to spaghetti-like process models.

We solve the described optimization problem by applying the Particle Swarm Optimization (PSO) algorithm [KE95]. PSO is a nature-inspired optimization algorithm for multivariate optimization problems, which only makes few assumptions about the problem being solved. For instance, it does not require the optimization function to be differentiable. Although the PSO algorithm does not guarantee that an optimal solution is found, our experiments show good results with reasonable effort and time (see Section 6.4.1.4).

The basic idea of PSO is inspired by the social interaction of individuals living in groups, exchanging information to support each other. PSO maintains a swarm of n moving particles $\vec{p}$, each maintaining its own position, a velocity and the best known position so far in the search space. Different from evolutionary algorithms, information about the best position any particle has found is shared among all particles. Initially, all particles $\vec{p}_0$ are randomly distributed in the search space and assigned an initial movement velocity $\vec{v}_0$. Particles move in time-discrete iterations and update their position based on

the inertia ω, the best known position $\vec{p}_{best}$, the global best position among all particles $\vec{g}_{best}$, the weighting factor c_k, and the social weighting factor c_s.

Definition 6.4 (Particle Swarm Optimization)**.** For each round of the PSO, the new velocity vector $\vec{v}_{n+1}$ is calculated as the follows:

$$\vec{v}_{n+1} = \omega \cdot \vec{v}_n + c_k \cdot r_1 \cdot (\vec{p}_{best} - \vec{p}_n) + c_s \cdot r_2 \cdot (\vec{g}_{best} - \vec{p}_n) \tag{6.7}$$

where r_1, r_2 are drawn uniformly at random from $[0, 1]$, updated in every iteration. The standard parameters for PSO are $\omega = 0.72984, c_k = c_s = 1.496172$ [CK02; BK07].

In our algorithm, we set the maximum number of iterations for PSO to 10 and introduce a new stopping criterion that terminates the search for a solution when the maximum fitness of 1 is reached for all process models.

In summary, the PSO algorithm finds appropriate parameter values for the clustering by maximizing the fitness of the underlying process models. As a result, sub-logs with traces of similar behavior on both, the control flow and the data perspective are returned.

6.4 Evaluation

This section presents the evaluation results of the hybrid feature set trace clustering algorithm on synthetic and real-life event logs. First, we evaluate the cluster quality of our trace clustering algorithm with respect to cluster and process mining evaluation measures (see Section 6.4.1.4). Second, we show the applicability of our algorithm to real-life event logs (see Section 6.4.2).

For the evaluation, the hybrid feature set trace clustering algorithm was implemented as a ProM plugin which is publicly available[1].

6.4.1 Synthetic Event Logs Evaluation

We generated a set of synthetic event logs to evaluate the cluster quality of our trace clustering algorithm and compared the results with the related work.

6.4.1.1 Experiment Setup

There is no comprehensive set of benchmark event logs in the literature, evaluating trace clustering approaches that focus on control flow and data perspective. Due to the lack of available datasets, we generated a set of synthetic event logs using the PLG2 [Bur16]. PLG2 is a tool that allows

1 Source code available at: `https://github.com/alexsee/HybridClusterer`

Model	# at	# tr	# dpi	max length	out-degree
P2P	14	16	6	9	1.14
Small	22	26	6	10	1.18
Medium	34	48	25	8	1.41
Large	44	56	28	12	1.27
Wide	56	75	39	11	1.34
Huge	36	53	19	7	1.47

Table 6.2: Process models used for generating the event log: Number of activity types (# at), number of transitions (# tr), number of variants (# dpi), maximal trace length, and out-degree.

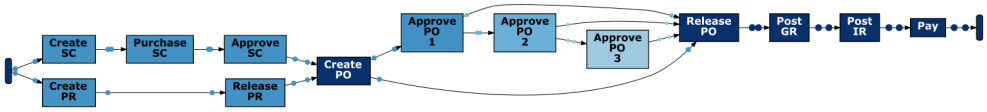


Figure 6.5: Causal net discovered by the heuristics miner of ProM showing the synthetic P2P event log.

generating event logs from process models by simulating the execution of the process model. We generated five different process models[2] [NSM16] which reflect different process complexities (i. e., varying number of activities, maximum depth, and branching factor): Small, Medium, Large, Huge, Wide, and a custom-designed model with human-readable activity names (see Figure 6.5 and Table 6.2). Our synthetic data set must contain process behavior that can only be distinguished by the combination of control flow and data perspective.

PLG2 only supports the modeling of simple case attributes. However, for this evaluation, the event logs must include dependencies of the case attributes among each other as well as to the control flow (see Figure 6.6). In order to generate representative case attributes, for each attribute $a \in \mathcal{A}$ a set of randomly attribute values $\mathcal{V}_a$ is generated, containing 20 different case attribute values. Each case gets the attribute values for each of the case attributes a sampled from $\mathcal{V}_a$. Causal relationships between case attributes among each other and of case attribute values are introduced by forcing certain combinations of case attribute values to occur more frequently than others, depending on the sequence of activities. Consequently, the event log contains sequences of activities with the same set of case attributes and values. It is noteworthy that due to the random relationship generation, different activity sequences may have the same attribute value pattern. As a result, generated event logs contain patterns that correlate with the control flow as well as the data perspective. For evaluating the clustering, a set of

2 Models are publicly available: `https://doi.org/10.7910/DVN/QBL1K0`

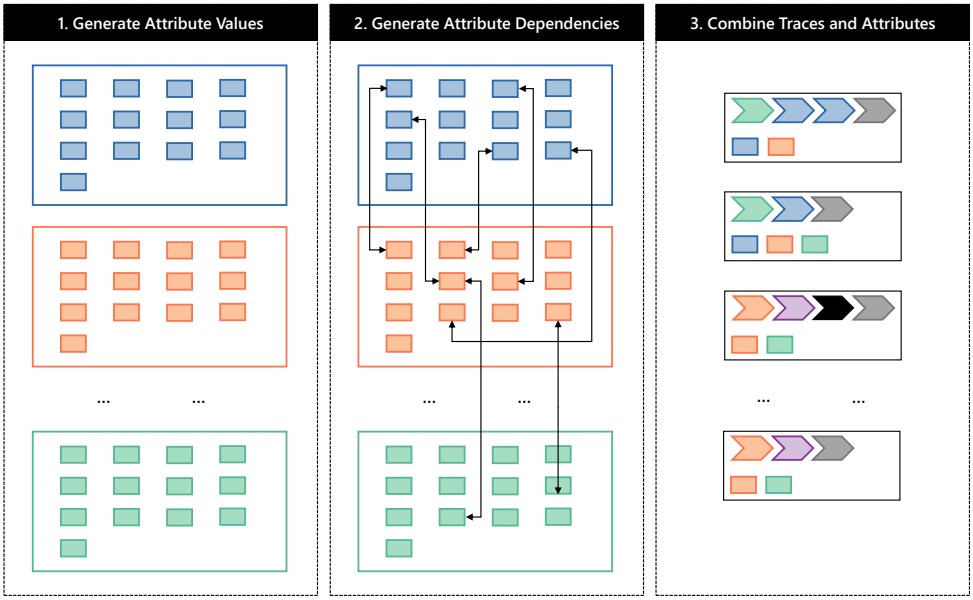


Figure 6.6: The synthetic event log generation containing different process behavior that considers the control flow and case attributes.

five different event logs with the same configuration but different patterns is merged that represent the ground truth data for the clusters.

In addition, we add a certain level of noise to the control flow in order to make the synthetic event logs resemble reality better. In total, a set of 288 event logs was generated from the six process models with different log sizes (1 000, 2 000, 5 000, 10 000), varying number of case attributes (5, 10, 15, 20) and three different noise levels (0.0, 0.1, 0.2).

6.4.1.2 Evaluation Metrics

We evaluated the quality of the trace clustering algorithms by measuring two types of quality metrics. First, we investigated the quality of the clusters regarding the ground truth. In particular, we report the average F1-BCubed and the Adjusted Rand Index (ARI) across all obtained clusters as introduced in Section 2.5.1.3. Second, we investigated the quality of the process models discovered for each of the obtained clusters. In particular, we report the average fitness and the precision across all obtained clusters as introduced in Section 2.4.3. The measures of each cluster are weighted according to the number of cases within a cluster. We use the PM4Py implementation[3] to compute fitness and precision.

3 Open source process mining library for Python; available at `http://pm4py.org/`

6.4.1.3 Trace Clustering Algorithms

We compare our multi-perspective trace clustering approach (HC) to the following related work:

BAG OF ACTIVITIES (BOA) [BA10] The bag of activities approach converts the activity occurrence of each case into a feature vector. The pairwise Euclid distance is stored as a similarity matrix which is used as the input for hierarchical agglomerative clustering.

LEVENSHTEIN EDIT DISTANCE (LED) [BA10] The Levenshtein edit distance approach compares cases by calculating the costs to modify the trace to another. The normalized costs are used to construct a similarity matrix which is, again, the input for a clustering algorithm.

ACTITRAC (ATC) [Wee+13] ActiTraC iteratively builds clusters of traces based on the fitness of the process models. Only traces that increase or preserve the fitness of the process model are assigned to the cluster. Instead of defining a similarity function, the algorithm tries to find an optimal assignment of traces to clusters, maximizing the overall fitness of each cluster.

CONTEXT-AWARE-CLUSTERING (CAC) [BA09] Context aware clustering is based on the Levenshtein Edit Distance (LED) approach, but also considers the context of the activities to determine the similarity of traces.

We consider two more approaches BOA+ and LED+ that are based on Bag-of-Activities (BOA) and LED respectively. In this case, all traces that occur less than 2-times in the event log are removed. Furthermore, the Flexible Heuristics Miner (FHM) [MDR17] is used as the baseline without clustering.

Due to the lack of an automatic parameter tuning approach for BOA, LED and Context-Aware-Clustering (CAC), the best results for the same or fewer clusters as provided by the Hybrid Feature Set Trace Clustering (HC) algorithm are considered. For ActiTraC (ATC) the standard settings are used: 80% stopping criterion for the frequency-based and MRA distance-base selective sampling.

6.4.1.4 Accuracy Results

This section presents the evaluation results of the synthetic event logs.

Weighted fitness. The measurements show that the weighted fitness is statistically significantly[4] different across all evaluated event logs depending on the applied method. Post-hoc analysis with Wilcoxon signed-rank tests and a Bonferroni correction showed that trace clustering statistically significantly increase the fitness of the process models compared to the FHM. The pairwise

4 Non-parametric Friedman test [Fri37]; $\chi^2(7) = 590.43$, $p < 0.001$

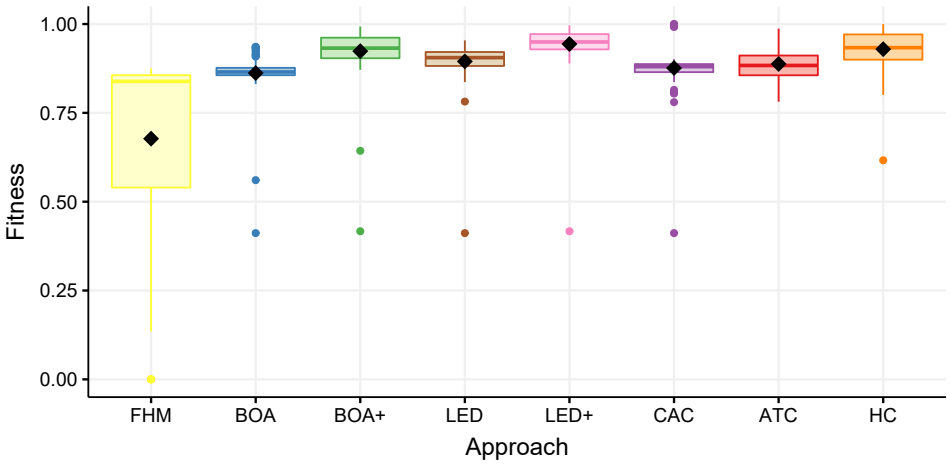


Figure 6.7: Process model fitness after trace clustering aggregated across all synthetic event logs with a noise level of 0.2.

comparison indicates no significant differences between CAC and BOA, as well as between ATC and LED. Figure 6.7 shows the weighted fitness across all synthetic event logs with a noise level of 0.2.

Precision. The precision measurements also show that the precision is statistically significantly[5] different across all evaluated event logs depending on the trace clustering method. Post-hoc analysis with Wilcoxon signed-rank tests and a Bonferroni correction indicated that all trace clustering methods, except for the LED, produced process models with significantly higher precision. There was no significant difference between HM and LED ($p = 0.078$). Figure 6.8 shows the precision measurements across all synthetic event logs with a noise level of 0.2.

F1-BCubed. The measurements of the F1-BCubed show that the quality of the clusters is statistically significantly[6] different across all evaluated event logs depending on the trace clustering method applied. Post-hoc analysis with Wilcoxon signed-rank tests and a Bonferroni correction, as well as the Nemenyi test as demonstrated in [Demo06], shows that HC performs significantly better than all other methods (see Figure 6.11). Based on the critical difference, we found that BOA and BOA+ lie in the same significance group. LED and LED+ lie in the same significance group and LED in the same as CAC. The F1-BCubed measurements aggregated across all synthetic event logs with a noise level of 0.2 are depicted in Figure 6.9.

Adjusted Rand Index. The measurements of the Adjusted Rand Index show similar results compared to the F1-BCubed measurements. The non-parametric

5 Non-parametric Friedman test [Fri37]; $\chi^2(7) = 613.09$, $p < 0.001$
6 Non-parametric Friedman test; $\chi^2(7) = 1458.3$, $p < 0.001$

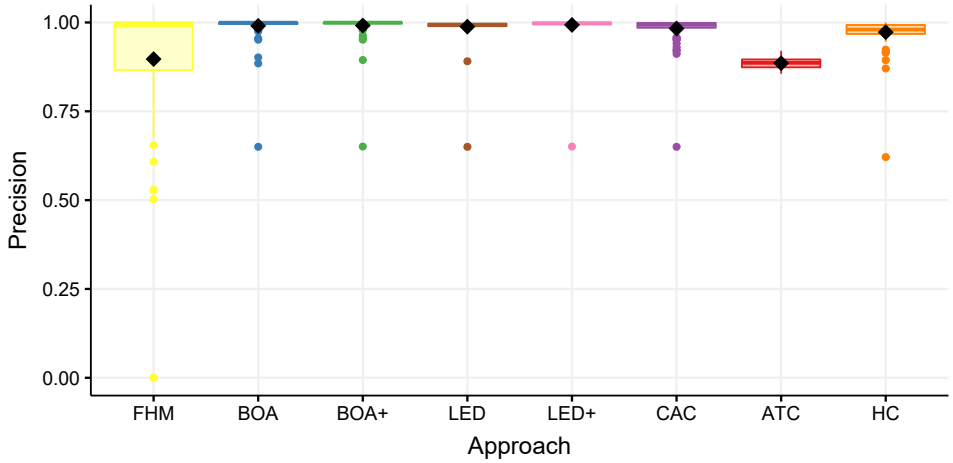


Figure 6.8: Process model precision after trace clustering aggregated across all synthetic event logs with a noise level of 0.2.

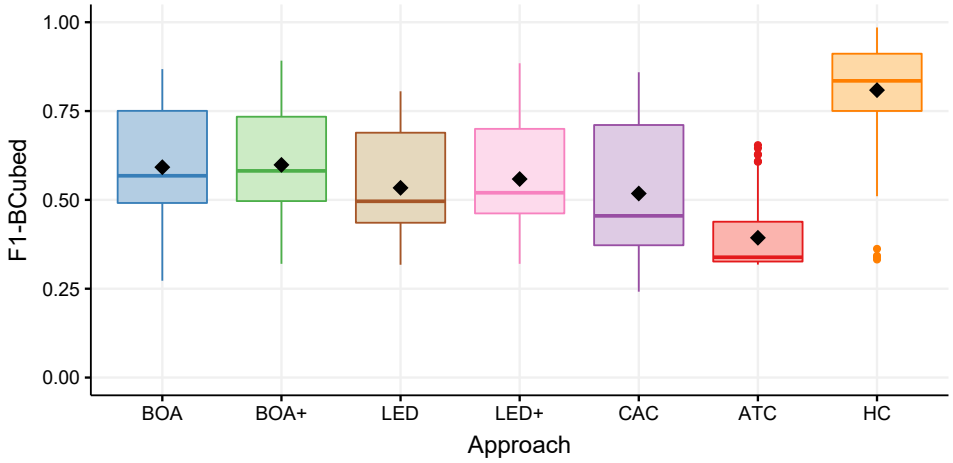


Figure 6.9: F1-BCubed results after trace clustering aggregated across all synthetic event logs with a noise level of 0.2.

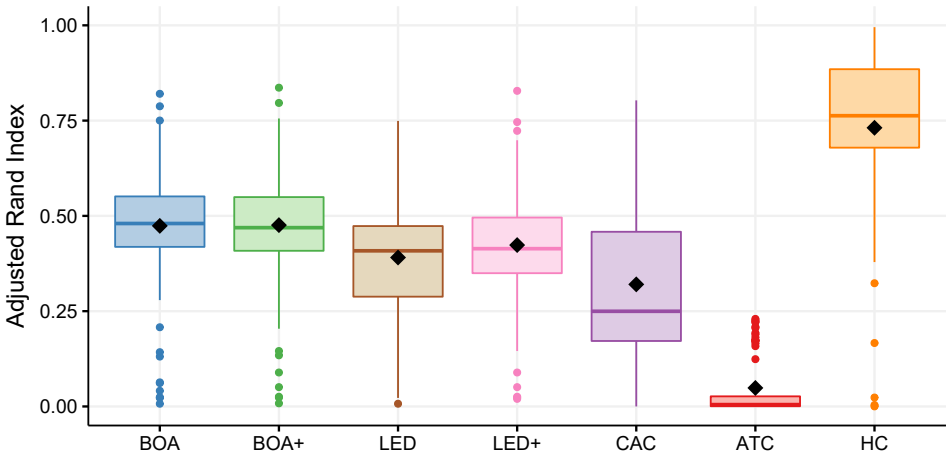


Figure 6.10: Adjusted rand index results after trace clustering aggregated across all synthetic event logs with a noise level of 0.2.

Friedman test indicates that the Adjusted Rand Index is statistically significantly[7] different depending on the applied trace clustering approach. Post-hoc analysis with Wilcoxon signed-rank tests and a Bonferroni correction indicate significantly better results of our HC algorithm for the Adjusted Rand Index. Furthermore, we found that there are no significant differences between the LED and the LED+, as well as between the BOA and BOA+ methods.

Discussion. The results of the evaluation show that trace clustering in general significantly improves fitness and precision of process models. We found that all compared distance and model-based trace clustering methods are useful to increase process model accuracy. However, we also found that the related work only focuses on the control flow and neglects other process perspectives. In our evaluation, our multi-perspective trace clustering algorithm is the only approach that combines the control-flow and the data perspectives. The existence of case attributes that are related to the execution behavior of the process is essential to the HC method. We only investigated synthetic event logs with such relationships. The results show significant improvements in the F1-BCubed and Adjusted Rand Index measures. Other comparable methods could show better performance if such relationships would not be present.

6.4.2 Real-life Event Logs Evaluation

This section presents the results of the evaluation of the real-life event logs.

7 Non-parametric Friedman test; $\chi^2(7) = 1520.9$, $p < 0.001$

	Fitness	Precision	F1-BCubed	ARI
FHM [MDR17]	0.8387	0.9617	-	-
Bag-of-Activities [BA10]	0.9245	0.9958	0.5819	0.4417
Bag-of-Activities+	0.9593	**0.9963**	0.5843	0.4414
Levenshtein [BA10]	0.9373	0.9936	0.5436	0.3871
Levenshtein+	**0.9653**	**0.9963**	0.5529	0.3973
CAC [BA09]	0.9286	0.9903	0.5295	0.3301
ActiTraC [Wee+13]	0.9460	0.9398	0.4082	0.0360
Hybrid clusterer	0.9652	0.9827	**0.8307**	**0.7628**

Table 6.3: Performance of the related work and the hybrid cluster approach with respect to process model and clustering evaluation; best values in bold typeface.

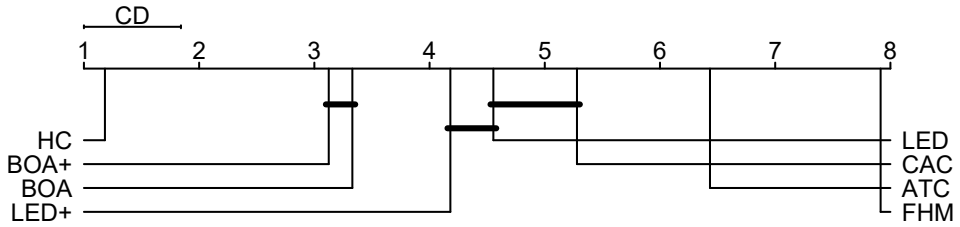


Figure 6.11: Critical difference diagram for all methods on all synthetic event logs; groups of methods that are not significantly different concerning the F1-BCubed (at $p = 0.001$) are connected.

Event log	Event log properties				Description
	# pi	# ev	# at	# dpi	
P2P	33 277	255 427	37	7 026	Procurement handling process
EV	1 434	8 577	27	116	Case handling system
HOSBILL	100 000	451 359	18	1 020	Hospital invoice billing
HOSLOG	1 143	150 291	624	981	Case handling in hospital
ROAD	150 370	561 470	11	231	Road traffic fine process
BPIC'19	251 734	1 595 923	42	11 973	Procurement handling process

Table 6.4: Overview of real-life logs: The number of process instances (# pi), number of events (# ev), number of activity types (# at) and the number of variants (# dpi).

6.4.2.1 Experiment Setup and Measurements

We used publicly available real-life event logs to measure the accuracy and simplicity of the resulting process models after applying trace clustering. Table 6.4 shows some basic statistics of the event logs used, originated from different environments and containing different types of processes. All event logs except for P2P are publicly available[8].

Similar to the first evaluation part, the same configuration (see Section 6.4.1.3) for the related work was used. Again, for BOA and LED the best results with the same or fewer number of clusters as HC are reported.

Each used event log originates from a real-life information system without any further information about how the process is intended to be executed. Different from the first evaluation part, specific evaluation metrics like the F1-BCubed and the Adjusted Rand Index (ARI) cannot be measured because the desired clustering result is unknown. Instead, the fitness and precision, as well as the number of clusters and the simplicity [Blu15] of the process models, are measured.

6.4.2.2 Accuracy Results

The measurements of the weighted fitness show that trace clustering outperforms the FHM except for the P2P dataset where BOA is slightly worse. This result suggests that in most cases a single process model is not sufficient for accurately representing the behavior of the event log. In particular, the event logs with a high number of traces get a low fitness value. For the P2P, HOSBILL, and BPIC'19 event log, the average weighted fitness across all subset clusters and provided by the HC outperforms all other methods. The event log HOSLOG shows a different result: the BOA+ and LED+ approach outperform all other clustering methods with a perfect fitness. This result

8 http://data.4tu.nl/repository/collection:event_logs_real

	P2P	EV	HOSBILL	HOSLOG	ROAD	BPIC'19
FHM	0.5435	0.7502	0.7630	0.2761	0.9112	0.4402
ATC	0.7716	0.9150	0.7625	0.3686	0.8942	0.6189
CAC	0.6384	0.9717	0.8413	0.1155	0.9548	0.5975
BOA	0.5337	0.7714	0.7823	0.4489	0.9805	0.5349
BOA+	0.6928	0.9617	0.7824	**1.0000**	0.8822	0.5755
LED	0.6249	0.9601	0.8401	0.5644	0.9513	0.5622
LED+	0.6999	**0.9850**	0.8460	**1.0000**	0.9821	0.5923
HC	**0.7723**	0.9788	**0.9693**	0.7949	**0.9876**	**0.7501**

Table 6.5: Results of the real-life event logs showing the average weighted fitness.

	P2P	EV	HOSBILL	HOSLOG	ROAD	BPIC'19
FHM	0.5800	0.4077	0.5000	0.0439	0.9763	0.8376
ATC	**0.7321**	0.8246	0.4007	0.0725	0.7464	0.6774
CAC	0.4485	0.9683	0.6068	0.1094	0.9928	0.7205
BOA	0.6678	0.9129	0.3507	0.1638	0.9860	0.6042
BOA+	0.5155	**0.9812**	**0.9922**	**0.9542**	0.9861	0.7649
LED	0.4632	0.8591	0.2675	0.2671	0.9152	0.6977
LED+	0.4787	0.9121	0.3914	0.9256	**0.9879**	0.8390
HC	0.6742	0.9450	0.6262	0.7327	0.8767	**0.8821**

Table 6.6: Results of the real-life event logs showing the average weighted precision.

can be explained by the characteristic property of the event log: it contains almost as many variants as traces. Due to the filtering of both methods, only a limited number of traces are considered. The CAC approach cannot properly deal with the HOSLOG event log and produces clusters with a lower average weighted fitness than the FHM. Table 6.5 summarizes the weighted average fitness results.

The measurements of the weighted average precision show mixed results. The HC approach outperforms the FHM except for the ROAD event log. However, other trace clustering methods provide better precision results for the event logs used. Table 6.6 summarizes the weighted average precision results.

6.4.2.3 Process Model Simplicity Results

The number of clusters in combination with the fitness results is an indicator of how good the trace clustering method can cluster the cases into clusters. In general, a low number of clusters and a high fitness are desired. For the CAC, BOA, BOA+, LED, and LED+ methods the number of clusters is reported

	P2P	EV	HOSBILL	HOSLOG	ROAD	BPIC'19
ATC	260	4	3	721	4	12
CAC	5	9	9	5	16	8
BOA	5	11	10	7	15	6
BOA+	5	14	3	4	13	3
LED	5	3	10	7	6	6
LED+	5	9	9	3	17	9
HC	5	14	10	7	18	9

Table 6.7: Results of the real-life event logs showing the number of clusters.

that is lower than the result of HC with the best fitness. Those methods do not provide an automatic approach to determine the optimal number of clusters. For the P2P and HOSLOG event log, the ATC approach generates a huge number of clusters although the fitness is comparable or lower than the other methods. The BOA and LED methods generate process models with a higher fitness for 3 out of the evaluated event logs with a lower number of clusters than the HC method. Similarly, the BOA+ and LED+ methods provide a higher fitness for 4 out of the evaluated event logs with a lower number of clusters. Table 6.7 reports the number of clusters of all compared methods across all event logs.

The results of the simplicity measure show a mixed result. For the P2P and the BPIC'19 event log, the ATC method generates simpler process models. However, the number of clusters is higher compared to the HC method, indicating that a higher fitness is only possible when cases are distributed to more clusters. The most simple process models for the EV event log are generated by the BOA method. For the HOSBILL, HOSLOG, and ROAD event log the HC method generates the most simple process models. The results suggest that trace clustering, in general, does not generate process models with lower complexity. Compared to the simplicity measurements of the FHM, the results are only slightly better. Table 6.8 reports the result of the average weighted simplicity of all compared methods across all compared event logs.

6.5 Discussion and Limitations

This chapter presented a multi-perspective trace clustering algorithm to split event logs into subsets of cases with similar behavior. However, there are some limitations and research directions for future work.

	P2P	EV	HOSBILL	HOSLOG	ROAD	BPIC'19
FHM	0.5773	0.5317	0.5273	0.5369	0.5254	0.5526
ATC	**0.5132**	0.5243	0.5405	0.5195	0.5151	**0.5290**
CAC	0.5640	0.5222	0.5235	0.5553	0.5160	0.5533
BOA	0.5675	0.5120	0.5205	0.5371	0.5213	0.5602
BOA+	0.5326	**0.4956**	0.5241	0.5156	0.5155	0.5462
LED	0.5653	0.5280	0.5204	0.5350	0.5246	0.5564
LED+	0.5371	0.5000	0.5181	0.5150	0.5095	0.5411
HC	0.5385	0.4972	**0.5163**	**0.5065**	**0.5071**	0.5468

Table 6.8: Results of the real-life event logs showing the average weighted simplicity.

6.5.1 Feature Space

A limitation of the approach is that it cannot include numeric case attributes because frequent itemset mining is not able to obtain dependencies from numeric attributes. However, numeric case attributes also influence the behavior of the process. With the current method of obtaining relationships between case attribute values, attributes with numeric values are not considered.

The current multi-perspective trace clustering approach only considers two process perspectives: control flow and data attributes. Although other process perspectives are also of high interest and reveal other process behavior dependencies, this has not been investigated in this thesis. For future work, it may be interesting to incorporate more than two process perspectives to identify different process behaviors.

Our approach uses a manually designed vector space representation to describe the control flow and the data attributes. In future work, the automatic training of these representations, e. g., the training of word embeddings, may significantly improve the overall performance of the multi-perspective trace clustering. Recent work [DBW18] has shown that word embeddings can be similarly applied to event logs of business processes.

6.5.2 Interestingness

Our trace clustering algorithm does not guarantee interesting clusters for the analyst because it is a fully-automated algorithm that adapts itself to the given event log. In particular, the algorithm has no notion about interestingness and, therefore, can not optimize the result towards interesting clusters. The result of the clustering algorithm is mainly balanced between the optimization of process model fitness and the similarity of case attributes. Although the event log may contain interesting case attribute relationships, these may not be discovered because the algorithm prefers process models fitness over case

attribute similarity. This behavior may lead to an clustering result that may not match the analyst's expectations because the analyst cannot control the influence of process model fitness. It could be interesting future work to give the analyst some control over how much influence the optimization of process model fitness should have.

6.5.3 Runtime Performance

Although we have restricted the Particle Swarm Optimization (PSO) to a maximum number of iterations and introduced a stopping criterion, the runtime performance of our approach could still be improved. Due to the current construction of the optimization algorithm, certain tasks are executed multiple times and are not parallelized. Parallelization and further optimizations may increase overall computational performance. It would be interesting to investigate more efficient process discovery and trace alignment methods, which are the bottleneck of our trace clustering approach.

6.6 Conclusion

This chapter introduced the Hybrid Feature Set Trace Clustering algorithm which clusters event logs into smaller sub-logs of similar process behavior to increase process model accuracy and reduce process model complexity. In summary, the main contributions of this chapter are:

1. A parameter-free multi-perspective trace clustering algorithm that combines the control flow and the data perspective of a process to generate clusters of similar process behavior.
2. A combined similarity function that allows balancing the contribution of both process perspectives considered, control flow and data.
3. An automatic parameter tuning that determines appropriate parameter values such as the weighting factor, the number of clusters, and the itemset support threshold by optimizing the fitness of the corresponding process models of each cluster.
4. Results from an experimental evaluation investigating the accuracy of the process models and adjusted rand index of the clusters. Furthermore, the practical feasibility of the approach.

Hybrid Feature Set Trace Clustering allows analysts to investigate the different process behaviors observed in an event log. It is a parameter-free algorithm, enabling the application to real-life event logs without any prior knowledge about the process inspected. As illustrated in Figure 6.12, this algorithm is used in the next chapter to obtain subsets of cases that contain different process behaviors. These subsets build the basis for guiding ana-

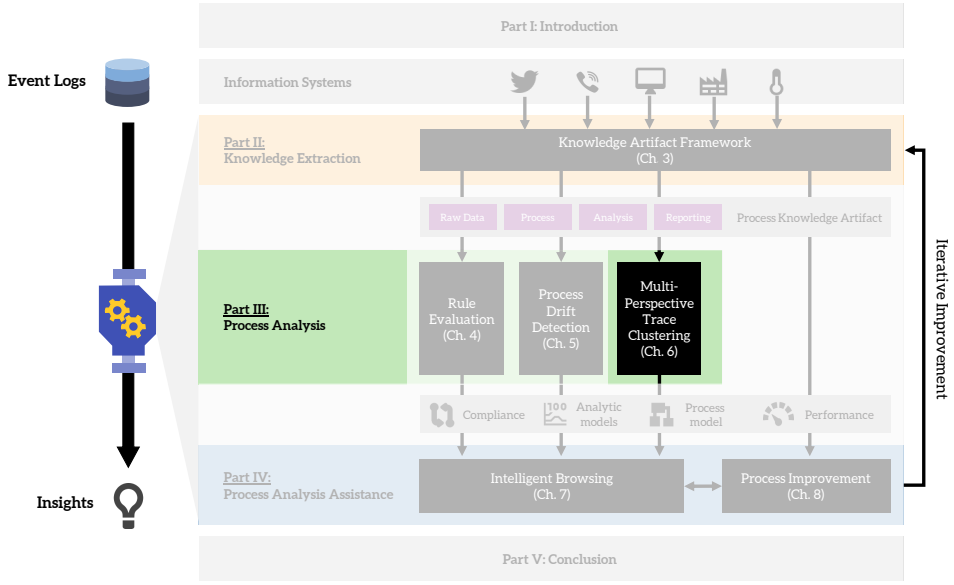


Figure 6.12: Overview of the chapter and contributions.

lysts towards interesting findings, improving the exploratory analysis for unknown event logs. The following chapter introduces ProcessExplorer, an interactive visual recommendation system, which integrates several chapters of this thesis to enable fast data exploration.

Part IV

Process Analysis Assistance

7

Intelligent Browsing

The previous three chapters investigated process analysis methods of targeted and exploratory process mining tasks. This chapter introduces ProcessExplorer, an intelligent and interactive visual recommendation system to enable fast data analysis and exploration of large and complex event logs. It closes the gap between autonomous targeted analysis tasks and the exploratory process discovery. ProcessExplorer is inspired by the workflow of analysts during the use of existing process mining tools to analyze discovered process models. The system extends existing process mining tools by introducing a recommendation engine that guides analysts towards automatically obtained findings in event logs. Particularly, it suggests interesting subsets of cases and computes insightful insights that analysts are typically interested in. We describe the ProcessExplorer approach that is based on the process knowledge artifact framework introduced in Chapter 3 and uses methods introduced in Part II of this thesis to enhance the visual exploration of large event logs.

This chapter is organized as follows: First, Section 7.1 gives a motivating introduction to exploratory analysis in the context of process mining. Next, related work is discussed (Section 7.2). In Section 7.3, the ProcessExplorer approach is introduced that guides analysts during the analysis of event logs. Section 7.4 describes the ProcessExplorer system that is used to evaluate the usefulness of the approach. Then, Section 7.5 describes the results of a user study with experts in the field. The chapter concludes with the limitations of the presented work, and potential future work (Section 7.6).

> **Publication:** This chapter is based on the following publications:
>
> Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. "Process Explorer: An Interactive Visual Recommendation System for Process Mining." In: *KDD Workshop on Interactive Data Exploration and Analytics* (2018).
>
> Alexander Seeliger, Maximilian Ratzke, Timo Nolle, and Max Mühlhäuser. "ProcessExplorer: Interactive Visual Exploration of Event Logs with Analysis Guidance." In: *Proceedings of the 1st International Conference on Process Mining - ICPM'19 - Demo*. 2019.

Alexander Seeliger, Alejandro Sánchez Guinea, Timo Nolle, and Max Mühlhäuser. "ProcessExplorer: Intelligent Process Mining Guidance." In: *Business Process Management*. Cham: Springer International Publishing, 2019.

Contribution Statement: I led the idea generation, implementation of the prototypes and performed the data evaluation. The student *Fabian Kaiser* implemented the prototype user interface for the first user study. The student *Maximilian Ratzke* implemented the prototype user interface for the second user study. *Timo Nolle, Alejandro Sánchez Guinea* and *Max Mühlhäuser* supported the conceptual design and contributed to the writing process.

7.1 Introduction and Motivation

The workflow of analyzing event logs using today's process mining tools is highly exploratory. It is driven by the knowledge of the analyst and guided by the desired analysis goal. Typically, the initial starting point of a process mining project is the discovered process model derived from a recorded event log of a Process-aware Information System (PAIS). Analysts inspect the discovered process model using various methods, including the visual inspection of the process model, the selection of subsets, and the computation of Process Performance Indicators (PPIs). Due to the massive growth of readily available event log data, the increasing complexity of the underlying process, and the flexible execution of processes in businesses, the visual exploration and analysis of processes incur several challenges.

As discussed in [SGA09], often only a highly complex and spaghetti-like process model is reconstructed from an event log which is hard and inefficient to analyze. Without a deep understanding of the corresponding process, the elicitation of interesting insights and trends is non-trivial. Consequently, analysts need extensive knowledge to correctly interpret the discovered process model [MRC07], investigate different subsets of cases, and calculate relevant PPIs before obtaining valuable insights and insightful visualizations.

We illustrate a scenario, which exemplifies the situation of an analyst may encounter with currently available process mining tools: *"Julia is an analyst who is interested in the process performance of the BPIC 2019 event log. The process discovery returns a spaghetti-like process model which reflects the actual behavior of the process. To simplify the view, Julia manually filters cases based on domain knowledge. For instance, she selects cases that start with a requisition and are of item type "subcontracting". Afterward, she computes the case duration of the subset and compares it with the case duration across all cases. For this particular selection, the case duration turns out to be significantly lower with 31.8d*

compared to 71.5d. Next, Julia considers a different selection and computes the case duration again." [See+19a] Many of the tasks Julia has to perform are executed manually, which is time-consuming and error-prone. Despite the increasing number of available process mining tools, today's tools only provide limited support and guidance, hampering efficient exploration and analysis.

In this chapter, we introduce ProcessExplorer, an interactive visual recommendation system for process discovery which is inspired by the workflow of analysts. ProcessExplorer extends the capabilities of existing process mining tools by providing automatic process analysis guidance. The system introduces two types of recommendations:

SUBSET RECOMMENDATION. ProcessExplorer suggests subsets of interesting cases that follow a similar process behavior observed in the event log. Subset recommendations are similar to case filters in existing tools but they are computed automatically from the event log. Typically, the selection of cases is a trial-and-error method that depends on the desired analysis goal and the discovered process model.

INSIGHT RECOMMENDATION. For each identified subset, ProcessExplorer suggests insight recommendations based on insightful PPIs characterizing the subset. Different from most existing process mining tools which let analysts manually design dashboards with statistical aggregations and charts, ProcessExplorer aims to automate this task.

ProcessExplorer applies diversifying top-k ranking with interestingness and support as the score to give analysts an extensive overview of the inspected process. The subset and insight recommendations are automatically computed from the event log only. ProcessExplorer does not make any assumptions about the underlying process and requires no parameter setup. Any event log that fulfills the requirements for process mining [IEE11; Aal11] can be used.

7.2 Related Work

This section introduces related work in the context of process analysis assistance. We divide the related work of this chapter in (1) exploratory data analysis, (2) data insights, and (3) interactive browsing in process mining.

7.2.1 Exploratory Data Analysis

Analyzing process models discovered from event logs is a highly exploratory data analysis (EDA) task [Aal11] that deals with the issue that users typically do not know the characteristics of an event log beforehand. In the data mining community, automatic visualization recommendations are proposed to support the user during data exploration. Self-organizing dashboards for visual

analytics are generated by VizDeck [Key+12], which automatically evaluates statistical properties of the dataset. Similarly, statistical and perceptual measures are used in Voyager [Won+16; Won+17] to automatically generate insightful data charts. Furthermore, Voyager introduces faceted browsing to quickly scan through the dataset. SEEDB [Var+15] evaluates the data columns and returns the most interesting visualizations. The "interestingness" is based on how large the data values deviate from a reference, e. g., a different subset or a different dataset.

Another research direction is to suggest the best type of visualization for datasets. AutoVis [WW08] is a data viewer that visualizes datasets appropriately based on the content, e. g., missing data, outliers, miscodes or anomalies. DeepEye [Luo+18] finds the best visualization based on supervised learning-to-rank machine learning and expert rules for better visual perception. A personalized visualization approach is VizRec [MVT16], which learns the preferred visualization based on the user's perception. SIDE [Lij+16] computes interesting projections of the dataset by letting the users express their interests or beliefs.

Research in the area of exploratory data analysis focuses on data in tabular form, making these approaches not easily applicable to event logs. Although event and case attributes can be transformed into a tabular form, methods must still consider the underlying process. To the best of our knowledge, no research combines methods of process mining with automatic exploratory data analysis to improve process discovery and exploration.

7.2.2 Data Insights

As an extension to exploratory data analysis, interesting insights can be provided automatically instead of manually exploring data dimensions or visual encodings. For instance, interesting patterns, such as exceptions [SAM98], high-variation patterns [Var+15], trends, and outliers [Che+02], from multi-dimensional data sets can be obtained using regression analysis [Che+02] or visual aspects [CYR09].

Several insights recommendation systems such as *Foresight*, *QuickInsights*, or *DBExplorer* have been proposed to provide useful insights automatically. Foresight [Dem+17] and QuickInsights [Din+19] evaluate statistical properties from categorical and numeric attributes. DBExplorer [SCJ16] improves the understanding of the data attribute characteristics and helps to query hidden attributes using conditional attribute dependency views. The system compares the attributes of the dataset and computes a view that groups similar information, revealing attribute dependencies. Data insights can be ranked by different aspects, such as dispersion, skew, outlier [Dem+17], or interestingness [ST96; GH06; Var+15].

A smart interactive drill-down approach is introduced in [JGP19], which discovers and summarizes interesting data groups. It allows exploring interesting parts of the dataset with fewer operations by suggesting a smart data selection. Similarly, Milo et al. [MS18] propose a next analysis step recommendation engine to improve data exploration. The next follow-up analysis step is predicted based on prior action recordings.

Although lots of work in the literature focuses on data insights, these techniques are not used in process mining to obtain interesting findings automatically. Similar to exploratory data analysis techniques, approaches have not been transferred and adapted to work with event logs. In addition, analysts in process mining have different interests and analysis goals than in data mining.

7.2.3 Interactive Browsing in Process Mining

There exists a range of academic (e. g., ProM, Apromore) and commercial (e. g., Celonis, Fluxicon Disco, PAFnow, Minit) process mining tools that support exploratory data analysis. These tools allow users to interactively inspect, analyze, and visualize event logs. ProM [Ver+11] is an open source academic process mining tool developed at the Eindhoven University of Technology. It comprises of a set of basic plugins for process discovery, conformance checking, and enhancement, and the package repository with over 1000 different plugins. Similar to ProM, Apromore [Ros+11] is an advanced process mining and modeling tool with rich capabilities.

Commercial tools like Fluxicon Disco (see Figure 7.1) or Minit[1] simplify process discovery by providing an easy to use user interface. Extensive visualizations of the process help the analyst to understand the actual behavior. Celonis or PAFnow Process Mining (see Figure 7.2) extend process discovery capabilities by integrating Business Intelligence (BI). BI enables "*structured analytics, visualization, predictive analytics, and Machine Learning and streaming capabilities*" [Hop19] in a data-centric fashion. Combining process mining with BI allows an in-depth and end-to-end analysis of a process. Analysts can explore the execution of a process from a data- and process-centric perspective, allowing to quickly investigate, for example, potential bottlenecks or compliance issues. Dashboards for PPIs allow analysts to quickly investigate the performance of the process.

VIT-PLA [Yan+16] supports the analyst by visually summarizing traces and generating data explanations from attributes using regression analysis automatically. A different approach introduces linguistic summaries [DW17]. The approach returns a list of statements that describe the event log, such as most cases containing a specific activity sequence having a high total duration

1 Commercial tool for process discovery; available at: `https://www.minit.io`

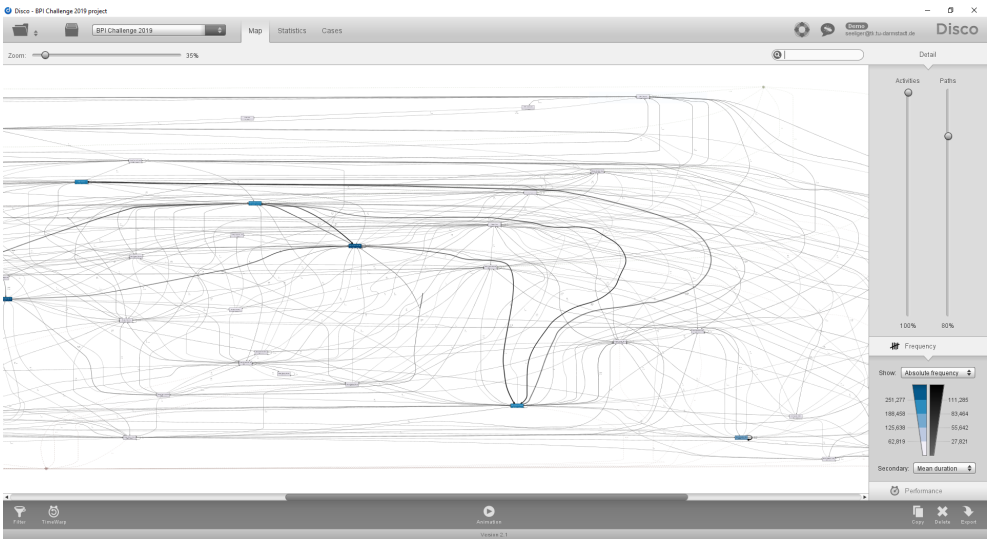


Figure 7.1: User interface of Fluxicon Disco showing the discovered process model of the BPI Challenge 2019 event log.

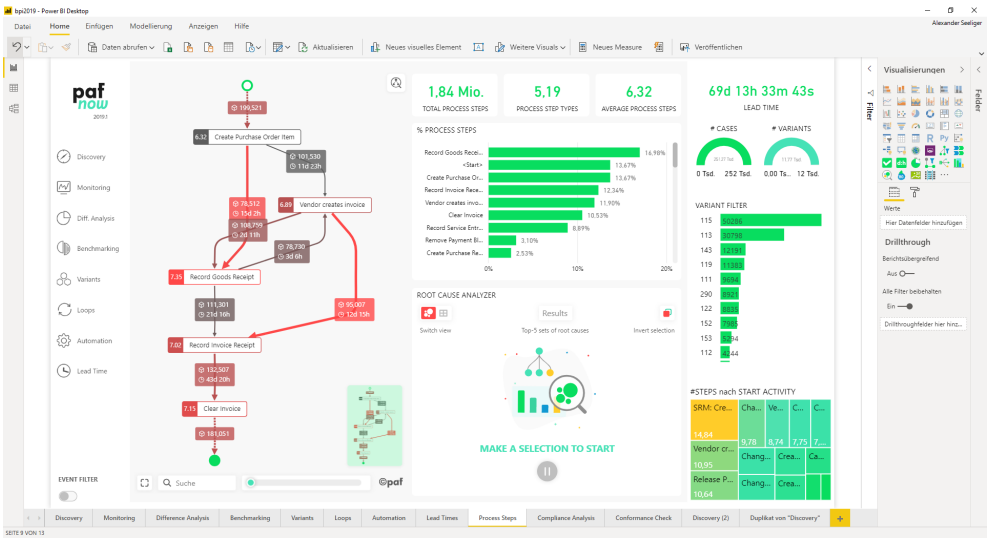


Figure 7.2: User interface of PAFnow Process Mining showing the process step evaluation tab of the BPI Challenge 2019 event log.

time. A process variant analysis using PPIs is introduced in [BSB17]. Here, the idea is to identify the key differences between the variants and extract their reasons. Similar work is introduced by Bolt et al. [BLA18] who annotate transition systems to obtain the key differences between variants. For each decision point, relevant case attributes and values are obtained using rule mining to determine the next follow-up activity in a process model.

None of the existing process mining tools combine manual interactive data exploration with automatic suggestions for interesting visualizations or insights. Analysts must manually search for interesting patterns, trends, and outliers. Related work that has been proposed in the data mining community cannot be easily transferred to process mining because the data structure and the analysis objectives are different. However, automatic suggestions in the exploratory process analysis are necessary to be able to deal with the increasing amount of data.

7.3 ProcessExplorer Approach

This section introduces the ProcessExplorer approach which is inspired by the workflow of analysts. The design of the approach is the result of a user study on the key requirements for building an interactive visual recommendation system for process mining. Although this user study was conducted before the approach was developed, it is presented in the evaluation section of this chapter (see Section 7.5.1).

Our approach provides two types of recommendations to the analyst by automatically inspecting the event log. The approach is divided into three main steps:

1. In a first step, ProcessExplorer automatically extracts subsets of cases with similar behavior using the multi-perspective trace clustering algorithm introduced in Chapter 6 (Section 7.3.1).
2. For each identified subset, ProcessExplorer automatically obtains only the most interesting PPIs by evaluating how much the subset deviates from a given reference (Section 7.3.2).
3. Each recommendation is ranked using diversifying top-k ranking for highlighting the most diversifying recommendations instead of showing similar subsets or PPIs (Section 7.3.3).

In the following, each step is described in detail.

7.3.1 Discovery of Subset Recommendations

In the first step, ProcessExplorer splits the event log into cases of similar behavior. Analog to the work of analysts who filter cases based on certain

criteria, we are interested in cases that follow the same process behavior. These subsets of cases build the basis for the subset recommendations suggested to the user by ProcessExplorer.

In order to obtain these subsets, we utilize trace clustering. Specifically, ProcessExplorer uses the multi-perspective trace clustering algorithm that was introduced in Chapter 6. Our algorithm combines two process perspectives, namely the control flow and data perspectives, to obtain subsets of cases that contain similar process behavior. This mechanism is derived from the work of analysts who typically combine multiple filters to obtain interesting subsets. For example, it may be interesting to only investigate cases of a specific department and that follow a specific activity order.

The multi-perspective trace clustering algorithm is parameter-free and automatically adapts to the given event log. This is especially important because ProcessExplorer is designed to automatically provide interesting insights from event logs.

7.3.2 Discovery of Insight Recommendations

In the second step, PPIs with the most interesting insights are obtained for each of the identified subset recommendations automatically. PPIs measure quantitative and qualitative aspects of the process, such as duration times, bottlenecks, and compliance violations, which are used by analysts to evaluate a process (cf. Table 7.1). ProcessExplorer alleviates the repetitive manual work for analysts caused by a large number of different PPIs that need to be evaluated.

Similar to the workflow of analysts who investigate different PPIs by visually preparing charts or descriptive results, ProcessExplorer computes all the different PPIs for each of the subsets and then compares them with the measurement of a reference. In our system, we allow analysts to select a different subset of cases, the entire event log, or a different event log as the reference. The basic idea of obtaining insight recommendations is derived from SEEDB [Var+15]. SEEDB judges the interestingness of visualizations by evaluating how large the deviation of the visualized data is from a reference. The larger the deviation, the more interesting it is for the analyst. Although there exist other characteristics that indicate interesting PPIs [GH06], analysts are particularly interested in these large deviations. For example, a high duration time for a subset of cases that differs from the average may violate service level agreements.

Our approach allows to define PPIs for two different categories:

- *Case-based Process Performance Indicators.* A case-based PPI is computed for each case within a subset individually. For example, the number

PPI		Description
Case	**Control-flow**	Directly/eventually followed by
		Loops
	Resource	Number/distribution of resources
	Data	Case/event attribute value
	Time	Duration of events/trace
	Function	(Co-) Occurrence of events
		Number of events
Subset	**Control-flow**	Start/End event distribution
	Resource	Attribute resources
	Data	Distribution of case/event attribute values

Table 7.1: Case- and subset-based process performance indicators (PPIs).

of activities, the total duration time, or the existence of an activity are case-based PPIs.

Each case-based PPI is defined as a measurement function as follows:

$$f : \mathcal{C} \to \mathbb{R} \tag{7.1}$$

- *Subset-based Process Performance Indicators.* A subset-based PPI is an aggregated measurement on subset level. For example, the distribution of resources within a subset are subset-based PPIs.

  Each subset-based PPI is defined as a measurement function over sets of cases as follows:

$$f : \mathcal{C}^* \to \mathbb{R} \tag{7.2}$$

We propose a set of basic PPIs which are depicted in Table 7.1. The proposed set covers a wide range of basic characteristics that are typically measured by analysts during the exploration. However, other more process-specific PPIs may be required to evaluate the process performance. Our approach is agnostic to the particular definition of the PPIs. Any PPI that can be described as a function according to the two categories (see Equations 7.1 and 7.2) can be used to extend the existing set of PPIs.

The interestingness of PPIs is obtained by performing statistical significance tests to check if the measured values of the subset differ significantly from the measured values of the reference set. In particular, the null hypothesis of the statistical significance test reads: the two measurements are drawn from the same distribution. If there exist some deviations between the investigated subset and the reference set, the null hypothesis is rejected for those PPIs that differ significantly. A significance level is set beforehand which is the

accepted probability that the null hypothesis is true but wrongly rejected. The p-value of a statistical significance test returns the probability that the observed result occurs under the null hypothesis. Based on the p-value and the significance level, ProcessExplorer decides to reject the null hypothesis or not. If the null hypothesis can be rejected according to the significance level, the corresponding PPI is considered to be interesting.

For each considered PPI, ProcessExplorer performs a statistical significance test to decide its interestingness. Depending on the type of the PPI, the two-sample Kolmogorov-Smirnov test [Mas51] is computed for case-based PPIs, and the Jensen-Shannon divergence [Lin91] is computed for subset-based PPIs. We use the two-sample Kolmogorov-Smirnov test because it is non-parametric, it does not assume that the data is sampled from a normal distribution, and it is also robust for various sample sizes. The Jensen-Shannon divergence is used because it is symmetric and always bounded [Lin91].

A typical issue with statistical significance testing is that even small deviations between the investigated sets of values can lead to a highly significant result [SF12]. A better quantitative measure for obtaining the magnitude of the deviation is Cohen's effect size [Coh88; Coh92]. For two measurement series x_1, x_2, Cohen's d effect size is defined as follows:

$$d = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{(s_1^2 + s_2^2)/2}} \text{ with } s_i^2 = \frac{1}{n_i - 1} \sum_{j=1}^{n_i} (x_{j,i} - \bar{x}_i)^2 \tag{7.3}$$

Cohen [Coh88] introduced comprehensive ranges to characterize the magnitude of the deviation: $0.2 < d \leq 0.5$ indicates a small effect, $0.5 < d \leq 0.8$ indicates a medium effect, and values $d > 0.8$ a large effect. Using this scale, analysts can immediately assess PPIs that are most deviating and have a high impact on the process behavior. ProcessExplorer indicates the severity of the deviation using a continuous color scale, from green (small effect; $d < 0.5$) to red (large effect; $d > 0.8$), based on the ranges introduced by Cohen.

Some of the PPIs have a strong correlation between each other, and they are likely equally significant. These correlations lead to redundant "insights", which would unnecessarily increase the total number of insights providing similar information. We simplify the list of identified insights by grouping correlating PPIs together using clustering. As input for the clustering, we use the Spearman correlation matrix which is computed for all relevant PPIs pairwise. The elbow method [HKP11] is used to identify the optimal number of clusters. The elbow method chooses the number of clusters by investigating the additional information gain that an additional cluster would provide. If the information gain drops, the number of clusters is chosen at this specific point.

7.3.3 Ranking of Recommendations

In the third step, ProcessExplorer ranks the generated subset and insight recommendations based on their relevance, yielding a ranked top-k recommendation list.

Each subset and insight recommendation is given a score that indicates its corresponding relevance. The score for insight recommendations is calculated as the product of effect size and case coverage within a subset. The coverage is defined as the proportion of cases within a subset for which the insight is applicable. Without considering the coverage of an insight, subset recommendations with a small number of cases are ranked higher although other subsets with more cases are more interesting. This intuition is also typically applied by analysts who investigate process issues that occur more frequently in the event log compared to issues that only occur once. However, if the effect size is large enough, even subsets with a low number of cases are still ranked higher. The score for the subset recommendations is the average score of all its insight recommendations.

A known issue of ranked lists is that they tend to prioritize similar items on top of the list. Furthermore, users tend to pay more attention to the top of the list instead of exploring lower-ranked items, leading to a low selection diversity. This behavior is known as the filter bubble effect [Ngu+14]. ProcessExplorer avoids this issue and provides a diversifying list of recommendations to the analyst by applying diversifying top-k ranking [QYC12] to the recommendation list. Diversifying top-k ranking investigates the similarity between the items and their score to return a ranked top-k list with the most diversifying items on top of the list. We use the same trace and case attribute similarity function as introduced for the multi-perspective trace clustering (see Section 6.3.1.3) to obtain the similarity between the subset recommendations. Based on the similarity between recommendations and their score, the algorithm returns a top-k list of diversifying recommendations which are used in ProcessExplorer.

7.4 ProcessExplorer System

The concepts of Section 7.3 are implemented into the ProcessExplorer system to illustrate and evaluate its usefulness. In this section, we describe the user interface and the architecture of the ProcessExplorer system.

7.4.1 User Interface

This section presents the graphical user interface of ProcessExplorer using the BPI Challenge 2019 [Don19] event log as the use case scenario. The event log contains recorded events from an information system that handles the

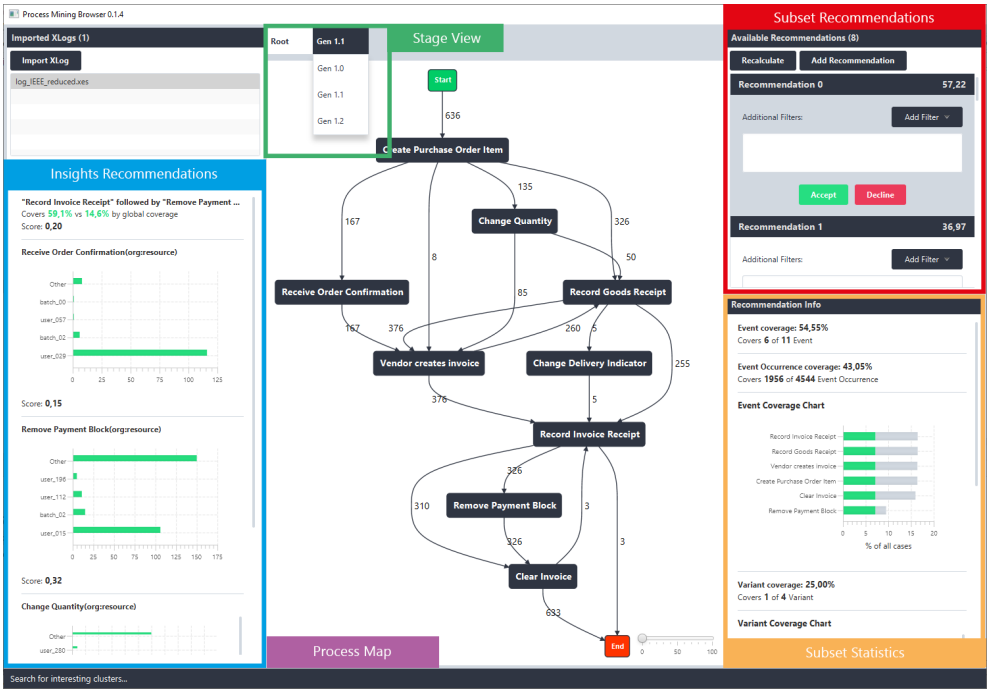


Figure 7.3: User interface of ProcessExplorer showing the subset and insight recommendations, the process map of the selected subset, the stage view, and the subset statistics.

procurement of goods in a multi-national company in the Netherlands. In total, the event log contains 1,595,923 events with 42 activities and 251,734 cases. This process involves 627 different resources, divided into 607 human users and 20 automated systems, also called batch users. For each purchase item, 16 different case attributes (e. g., item type, source, company, vendor) are recorded.

Figure 7.3 shows the overall user interface of ProcessExplorer. In the following, each of the user interface components is described in detail.

7.4.1.1 Subset Recommendations

After an event log is loaded into ProcessExplorer, it computes the subset recommendations. The list of the top 10 most relevant and interesting subset recommendations is shown to the user, depicted in Figure 7.4. Each recommendation is assigned a score that indicates its relevance and interestingness for the user. The list is sorted by the score and filtered using top-k diversifying ranking. The user can manually create new recommendations or modify existing recommendations by adding additional filters:

- *Variant filter*: The variant filter allows the user to select a specific process variant, i. e., all cases that follow a specific order of activities.
- *Start and end activity filter*: The start and end activity filter allow the user to select all cases that start or end with a specific activity.

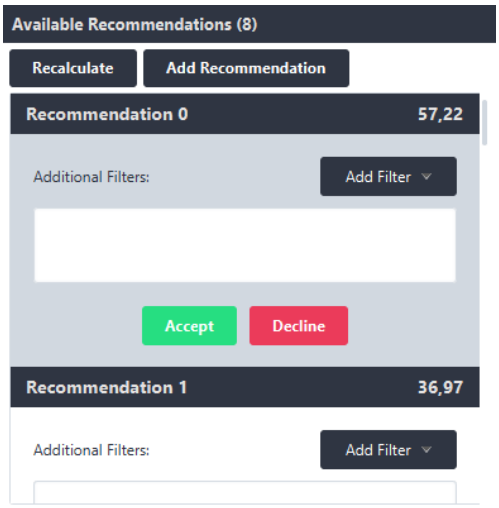


Figure 7.4: Subset recommendations of the BPI Challenge 2019 event log.

- *Happy path filter*: The happy path filter allows the user to select all cases that follow the most occurring process variant.

Subset recommendations can be accepted, which filters the event log according to the subset recommendation, or declined, which removes it from the list.

7.4.1.2 Subset Statistics

After the user selected a subset recommendation, the subset statistics component shows some basic statistics about the cases selected by this recommendation. The subset statistics component is shown in Figure 7.5. It shows the activity distribution, the variant distribution, the number of selected traces, and the number of selected transitions. For the example event log investigated, the first recommendation selects 6 out of 11 events and covers 1956 event occurrences.

7.4.1.3 Insight Recommendations

The ranked list of insight recommendations of an accepted subset recommendation is depicted in Figure 7.6. The visual presentation of an insight recommendation depends on the type of the insight. Trace-based PPIs are presented as a textual description that describes the deviation identified. Cluster-based PPIs are visually presented as a bar chart that shows the distribution of values. The subset recommendation considered in Figure 7.6 contains six insight recommendations. We highlight two of them:

- The first insight recommendation indicates a higher occurrence of the transition "Record Invoice Receipt" and "Remove Payment Block". In the subset, 59.1% of the cases follow this transition, compared to 14.6% in the event log overall.

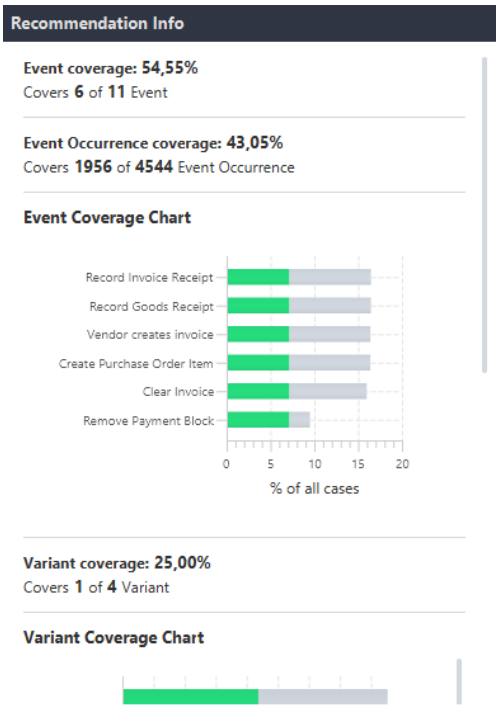


Figure 7.5: Statistics of a selected subset recommendation of the BPI Challenge 2019 event log.

- The second insight recommendation shows the distribution of resources for the activity "Receive Order Confirmation". The distribution visualizes that "user_029" executed this activity in most cases.

7.4.1.4 Process Map

Similar to other process mining tools, the process map visually shows the relationships between the activities observed. It allows analysts to inspect the directly-follows relation of activities. The user can navigate through the map, hide activities and transitions according to the relative occurrence, and see the number of cases that followed a specific transition. Figure 7.3 shows the process map after a suggested subset recommendation was accepted by the user.

7.4.1.5 Stage View

ProcessExplorer introduces stage views to simplify the navigation between the subset recommendations. Each time the user accepts a subset recommendation, a new stage is generated that stores the selected cases and the computed insight recommendations. Stages are organized as a hierarchical structure such that each refinement of a selection results in a new hierarchy level. For each stage, subset and insight recommendations are computed, so that recommendations can be successively refined. Figure 7.8 shows the stage view navigation after accepting three subset recommendations.

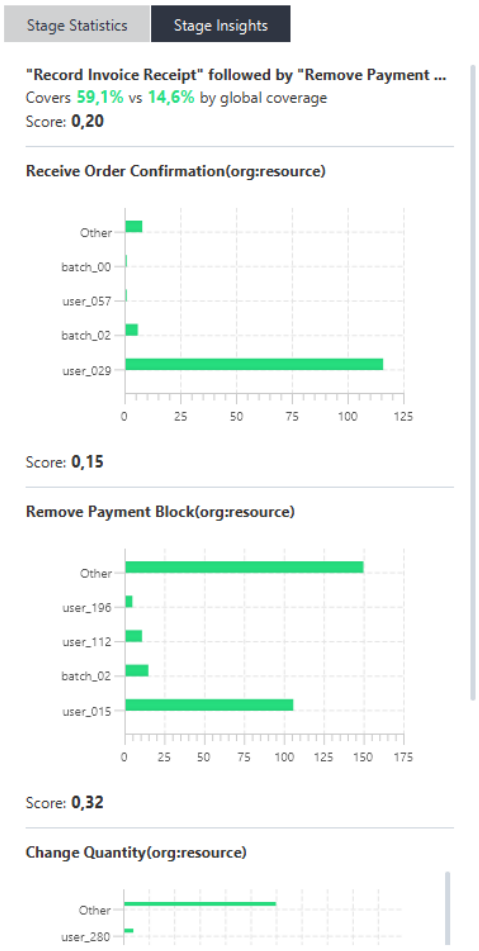


Figure 7.6: Insight recommendations of a selected subset recommendation of the BPI Challenge 2019 event log.

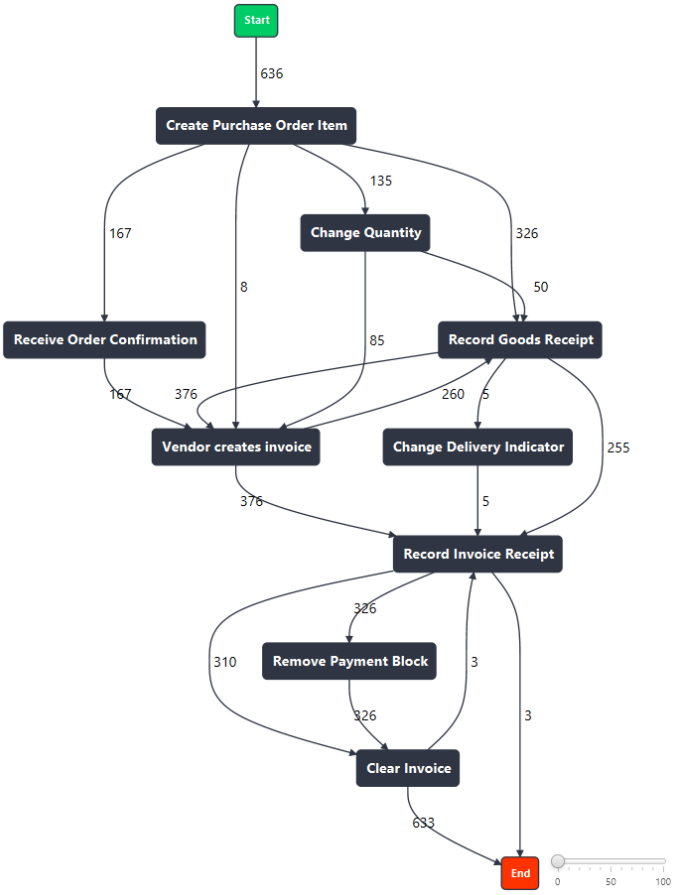


Figure 7.7: Process map of ProcessExplorer showing a subset recommendation of the BPI Challenge 2019 event log.

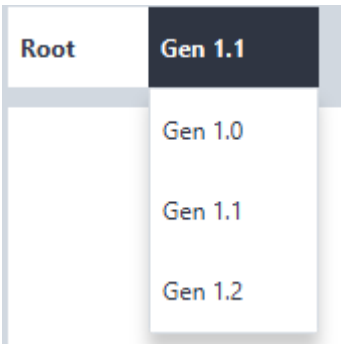


Figure 7.8: Stage view navigation of ProcessExplorer.

7.4.2 Architecture

In this section, we present the details of the reference architecture used to evaluate the ProcessExplorer. Figure 7.9 shows the architectural overview of the ProcessExplorer system.

7.4.2.1 Management

The recommendation framework of the ProcessExplorer system consists of three main management components which handle the event log data (*EventLogManager*), the recommendations (*RecommendationManager*), and the stage view navigation (*StageManager*). The components are designed to be extensible such that different event log formats, a different stage view management, and additional subset and insight recommendation engines can be integrated in the future.

ProcessExplorer supports the IEEE XES [Ver+11] event log format and uses the OpenXES implementation with the XESLite extension [Man16] (*InMemoryStore*) to load event logs. Event logs are stored in the *EventLogData* (see Figure 7.9 middle top) object in the form of an *XLog* object, which refers to the standard implementation of OpenXES. *EventLogData* additionally stores the basic statistics (i. e., the number of cases, the number of activities, the activity distribution, the number of variants, the variant distribution, the number of transitions, and the transition distribution) of the log that are shown to the user during the exploration.

The navigation in ProcessExplorer is organized in a hierarchical structure, the stage views. For managing the stage views, the *StageManager* (see Figure 7.9 center) stores a history of all stages visited. Each stage refers to a specific event log and can contain other stages that refine the case selection. ProcessExplorer always has an active stage that reflects the currently selected cases and the recommendations shown to the user. If the user decides to change the active stage, the *StageManager* retrieves the subset and insight recommendations from the *RecommendationManager*.

7.4.2.2 Recommendations

The recommendations are organized and managed by the *RecommendationManager* (see Figure 7.9 middle bottom). A recommendation consists of a case selection, i. e., the cases that should be selected by the recommendation, and the insights, i. e., the interesting findings for the selection. Each time a recommendation is requested, ProcessExplorer checks if the recommendations for this specific set of cases are already computed and returns the cached result, otherwise, it calls the *RecommendationEngine*. Recommendations are stored in the *Recommendation* object, which specifies the case selection using filters, and the insight recommendations.

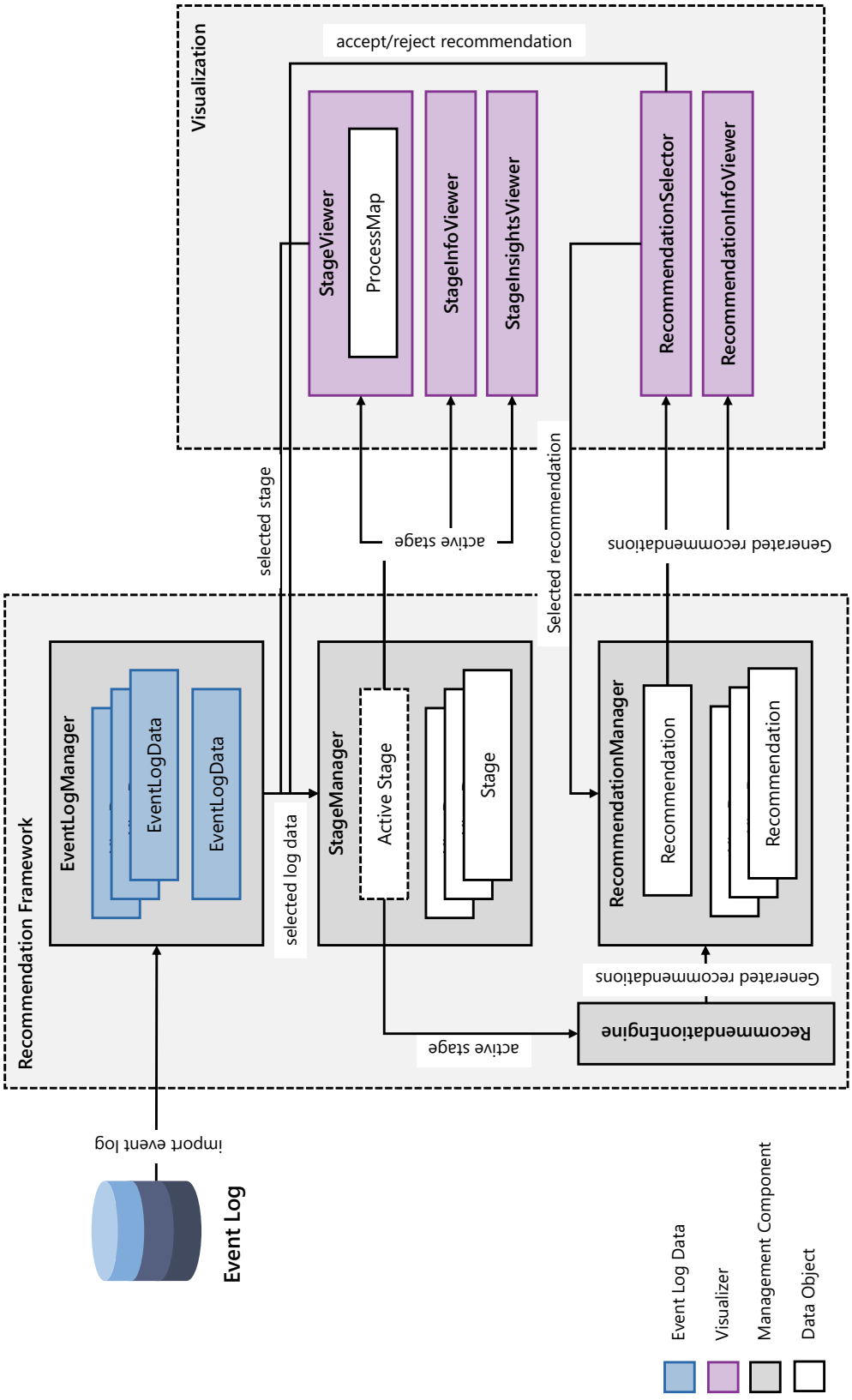


Figure 7.9: Architectural overview of the ProcessExplorer system [See+19b].

In the prototype implementation of ProcessExplorer, the multi-perspective trace clustering algorithm (see Chapter 6) is used for the subset recommendations, and statistical significance testing is used to obtain the insight recommendations. Due to the extensible architecture, other implementations for the recommendation engine can also be implemented.

7.4.2.3 Visualizations

Separate components are responsible for visualizing the event log, the log statistics, and the recommendations (see Figure 7.9 right grey-shaded box). The *StageViewer* is responsible for visualizing the active stage. In ProcessExplorer the process is visualized as a process map, implemented by the *ProcessMap* object. The *StageInfoViewer* shows the basic statistics of the active stage. The *StageInsightsViewer* is responsible for visualizing the list of insight recommendations of the active stage. Insights recommendations can be either visualized by a bar chart or by a textual representation.

The list of subset recommendations is visualized by the *RecommendationSelector*. If the user decides to select a specific subset recommendation, the *RecommendationSelector* reports the selected recommendation to the *RecommendationManager*. The *RecommendationSelector* also allows the user to customize recommendations. These customizations are reported back the *RecommendationManager*. Subset recommendations specify which cases are selected, hence basic statistics can be shown to the user. These statistics are visualized by the *RecommendationInfoViewer*.

All visualization components are separated from the underlying implementation of the recommendations such that other types of visualizations can be explored without changing the actual computation of the recommendations.

7.5 Evaluation

This section presents the evaluation results of two user studies regarding the usefulness of analytic guidance during the exploration of event logs. The first study (see Section 7.5.1) investigates the key requirements to build an interactive visual recommendation system for process mining using an early prototype. It is noteworthy that the first user study was conducted before the ProcessExplorer approach was designed. The second study (see Section 7.5.2) evaluates the usefulness of the ProcessExplorer system.

7.5.1 User Study: Identify Key Requirements for Analytic Guidance

Since analyzing event logs using process mining techniques is largely a manual exploratory task to date, this within-subject design study investigates how systematic analytic guidance can be provided to the user. The goal of

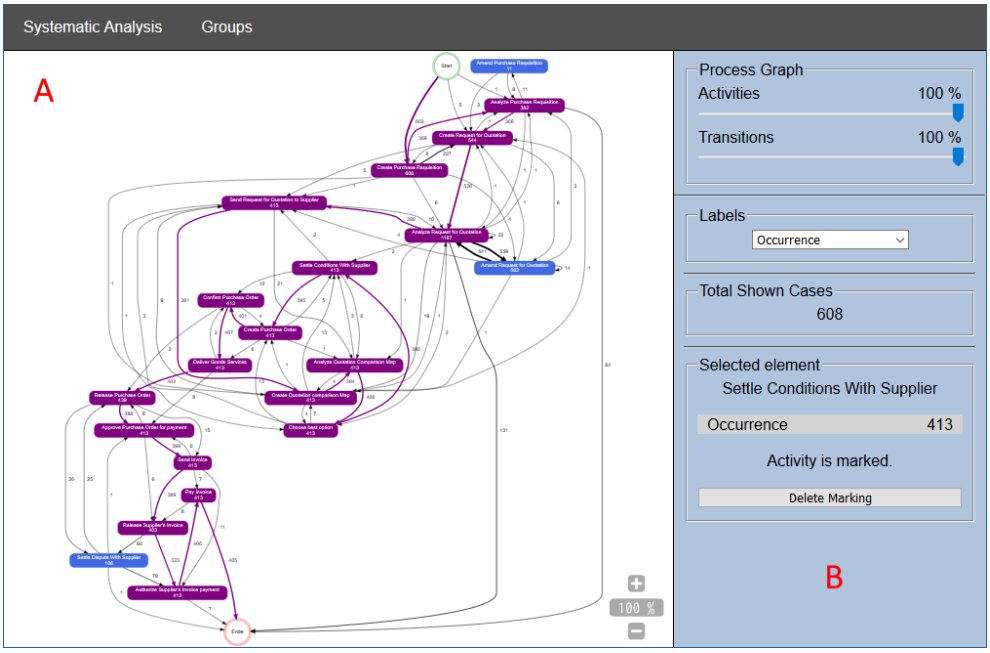


Figure 7.10: User interface of the early prototype of ProcessExplorer. Section A shows the process map and section B shows the activity and transition selection panel.

this user study is to evaluate which kind of guidance is useful for the analysis of event logs using process mining techniques. It specifically investigates the analytic guidance of process discovery and analysis techniques to identify potential process issues.

7.5.1.1 Setup

An early prototype (see Figure 7.10) of ProcessExplorer was shown to five process mining consultants. Study participants were familiar with existing state-of-the-art tools such as Fluxicon Disco, PAFnow Process Mining (all participants), QPR ProcessAnalyzer (4 participants), Celonis, and ProM (2 participants each).

The prototype contained three modes which refer to the degree of analytic guidance:

NO GUIDANCE. The first mode contains no guidance at all but manual filtering such as the activity, transition, cases, and variant filter. The tool offers the basic process discovery features that are commonly used by users.

STAGE VIEWS AND MARKINGS. The second mode introduces the stage view which allows the user to refine the filter selection stepwise. The user can mark certain activities and transitions (see blue highlighted activities in Figure 7.10) such that these markings can be hidden or shown only. This feature allows users to hide already investigated aspects.

SEMI-AUTOMATIC GROUPING. The third mode further introduces the grouping feature which semi-automatically provides the user with certain sets of cases. Markings introduced in mode two are now automatically added by selecting a group.

In the user study, we used the publicly available BPI Challenge 2017 [Don17] event log. First, all participants were asked to get familiar with each of the evaluated guidance modes, beginning with no guidance and ending with the semi-automatic grouping. The participants were requested to investigate and analyze the event log with the goal of obtaining valuable process knowledge. After participants explored the event log with a specific mode, they were asked to fill out a post-stage questionnaire to rate their explicit experience and preference with the interface and guidance. Finally, the participants had to fill out the User Experience Questionnaire (UEQ) [LHS08] which consists of 5 scales: *perspicuity*, *efficiency*, *dependability* (pragmatic quality), *stimulation*, and *novelty* (hedonic quality). For each of the scale, the UEQ returns a score from -3 (horribly bad) to $+3$ (extremely good).

7.5.1.2 Results

The responses of the post-stage questionnaire indicate that the stage view with the ability to navigate between different subsets of cases is beneficial and useful. The navigation paradigm was positively rated for the exploration of large event logs. However, two participants had the feeling that they took longer to analyze the given event log due to the increased navigation choices. Changing the selection of cases required the users to switch to a different view which was rated negatively.

The group feature which provided the ability to select a set of cases was rated positively. At the same time, the information displayed to the user was not sufficient to decide which group to select. Participants commented that providing some basic statistical information or a preview of the cases in the subset would increase the usefulness of the grouping. One participant argued that the top-down approach makes it difficult to obtain the really important parts of the process. For this reason, a recommendation feature that suggests interesting subsets of cases was proposed. These subsets would then be inspected one-by-one.

Figure 7.11 shows the results of the UEQ for the first and the second user study. According to the results of the first user study, the aspects stimulation and novelty got the highest average scores with 1.2 and 1.1 which refers to a positive evaluation. Attractiveness, efficiency, and dependability got a neutral score of 0.67, 0.6 and 0.15. The aspect perspicuity got a negative score value of -0.55 but in the interval, which is considered as neutral range. The results of the UEQ do not provide conclusive evidence that using the early prototype improves the analysis experience. Detailed comments from the

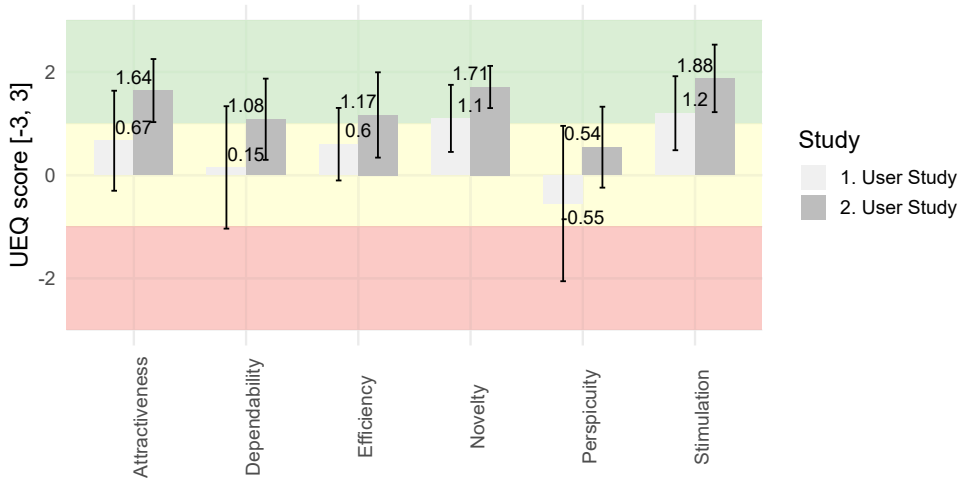


Figure 7.11: Results of the UEQ for the requirements study and for the final prototype. Range from -3 (horribly bad) to 3 (extremely good).

study participants indicate a steep learning curve of the used prototype, indicating that more guidance and a better user interface is required. Still, the idea of providing a better-guided analysis experience was rated innovative and promising.

7.5.2 User Study: Evaluation of Usefulness

This section presents the results of the user study performed to evaluate the usefulness of the ProcessExplorer system and the underlying concepts. The focus of this study is to asses how useful the subset and insight recommendations are for the exploration of large event logs.

7.5.2.1 Setup

In a user study workshop, ProcessExplorer was shown to six process mining experts who differ from those of the first user study. In total, the workshop took 60 minutes and was divided into two parts. The first part introduced the ProcessExplorer system to the study participants, showing the basic implemented guidance and recommendation features. For the workshop, an event log containing activities of a procurement handling process was used. There was only little to no further explanation required because the underlying process was well known. In the second part, participants were able to explore all the guidance features, i. e., subset recommendations, insight recommendations, and ranking of recommendations, of the ProcessExplorer approach on their own. Finally, the participants were asked to express their explicit opinion of the system.

The usefulness of ProcessExplorer was evaluated by applying the Technology Acceptance Model (TAM) [Dav85]. The TAM assess the potential

Question	Cluster	Mean	σ
1. Using ProcessExplorer improves the quality of the work I do.	A	2.60	0.49
6. Using ProcessExplorer improves my job performance.	A	1.17	2.11
8. Using ProcessExplorer enhances my effectiveness on the job.	A	2.40	0.80
3. ProcessExplorer enables me to accomplish tasks more quickly.	B	2.40	0.80
5. Using ProcessExplorer increases my productivity.	B	1.67	2.21
7. Using ProcessExplorer allows me to accomplish more work than would otherwise be possible.	B	1.80	1.94
4. ProcessExplorer supports critical aspects of my job.	C	1.33	2.21
9. Using ProcessExplorer makes it easier to do my job.	C	2.60	0.49
2. Using ProcessExplorer gives me greater control over my work.	D	2.20	0.75
10. Overall. I find the ProcessExplorer system useful in my job.	NA	2.80	0.40

Table 7.2: The questions and results of the Technology Acceptance Model (TAM) usefulness estimation.

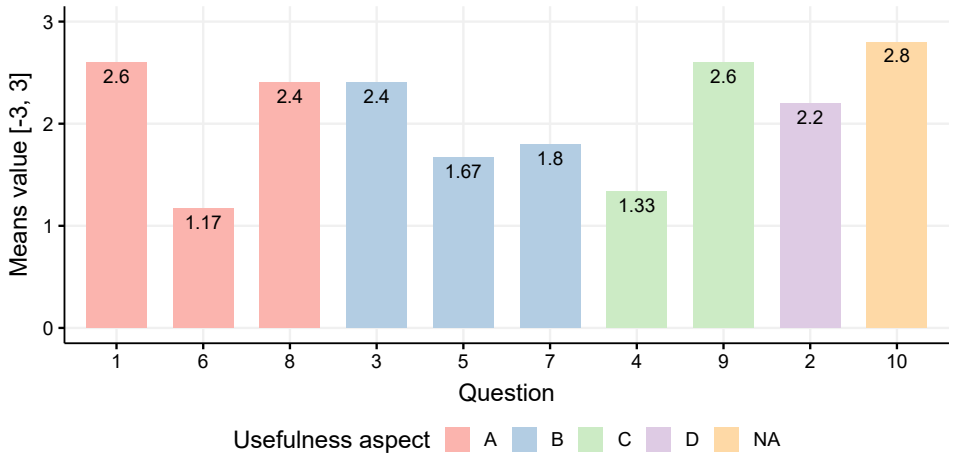


Figure 7.12: Overview of the TAM usefulness estimation for ProcessExplorer according to the question clusters which scales from -3 (strong disagree) to +3 (strong agree).

acceptance of a technology by end users using a quantitative evaluation in four different usefulness aspects:

(A) Job Effectiveness

(B) Productivity

(C) Importance of the system to the user's job

(D) Control over Job

The questionnaire consists of 10 statements (see Table 7.2) each assigned to one of the usefulness aspects. Study participants were asked to rate the statements on a scale of -3 "strongly disagree" to +3 "strongly agree".

Although the TAM is also capable of estimating the ease of use of a technology, we did not evaluate this aspect. Due to the limited time study participants spend with the system, results would have been misleading because getting familiar with the unknown tool may take more time. It is also a particular challenge to compete with existing process mining tools, which contain a large number of features expected by analysts. In this study, we were more interested in the usefulness of the concepts and not primarily on the ease of use of our prototype. Ideally, concepts of ProcessExplorer are integrated and evaluated in an existing process mining tool.

The user experience of the ProcessExplorer system was compared with the early prototype. Additionally, the UEQ was also be evaluated.

7.5.2.2 Technology Acceptance Model Results

According to the TAM, ProcessExplorer received an overall positive mean usefulness score of 2.8 ($\sigma = 0.40$). Specifically, each of the usefulness aspects received a positive mean score. According to the assessment of participants, the system improved the quality of the work they do (2.6; $\sigma = 0.49$) and enhances the effectiveness of their job (2.40; $\sigma = 0.80$). With respect to job improvement (A) the participants scored the system with 1.17 ($\sigma = 2.11$) which indicates that not all participants agreed on this statement. This result can be explained by the short evaluation time of the system, which may have prevented participants from becoming fully familiar with the system. The results of the job effectiveness aspect show that ProcessExplorer helps analysts during their analysis work of event logs. Especially the quick exploration of large event logs has a positive effect on the job effectiveness.

With respect to the productivity aspect (B), the study participants agreed that ProcessExplorer enables them to accomplish their tasks more quickly (2.40; $\sigma = 0.8$). They also agreed that the system allows the participants to accomplish more work than would otherwise be possible (1.80; $\sigma = 1.94$), and increases their productivity (1.67; $\sigma = 2.21$). Due to the automatically generated suggestions, users were able to explore interesting aspects more quickly than with conventional systems. Study participants also agreed that ProcessExplorer makes it easier for process analysts to do their job (2.60; $\sigma = 0.49$), and some of them agreed that it supports critical aspects in their job (1.33; $\sigma = 2.21$). The lower level of agreement to support critical aspects in their job may be because the system has not been told on which analytical objective it should focus. The statement that ProcessExplorer gives greater control over the work got a high agreement (2.2; $\sigma = 0.75$).

7.5.2.3 User Experience Questionnaire Results

The results of the UEQ show that all five scales improved compared to the requirements user study. The aspects "Attractiveness", "Novelty", and "Stimulation" were voted clearly the above average in the second study. These results confirm that participants found the significantly different user interface better than the one of the previous early prototype.

7.5.2.4 Free-text Comments

The questionnaire also allowed participants to give individual feedback, which is explained in this section.

On the one hand, most of the participants liked the general idea of getting analysis assistance during their work with the system, which improved their first exploration of an unknown dataset. The idea of automatically generating subsets of cases and presenting them as recommendations was described as "very innovative" (P1) and "super interesting" (P4). Likewise, one participant

commented, that ProcessExplorer is "a very useful tool for gaining quick control over unknown data" (P5).

On the other hand, the study also revealed some weak aspects of ProcessExplorer. One participant (P6) was not convinced about the insight recommendation and argued that these were not useful for his work. In our post analysis of the workshop, we investigated the real-life event log provided by the Business Process Intelligence Challenge (BPIC) 2019. Here, we found several insights such as rework activities and high duration times, that were also found by the participants of the challenge. However, it may be necessary to conduct more research in this context to get a better understanding of which insights are relevant, and how they should be presented to the user. Still, the subset recommendation was seen "as a real added value to process mining" (P6). Two other participants found that the user interface (P2, P4) was too overloaded with all the information shown at the same time.

7.6 Discussion and Limitations

This chapter presented ProcessExplorer, an interactive visual recommendation system for process mining, to enable fast data exploration. However, there are some limitations and research directions for future work.

7.6.1 Recommendations

The current approach considers two types of recommendations in the area of process discovery. Potential future work could investigate activity recommendations as well as suggesting different types of visualizations. Also, other process mining areas such as conformance checking and process enhancement could benefit from providing certain guidance to the analyst.

The user study with experts showed that the insight recommendations are not always of high value because they are independently computed without respecting the analysis goal. Furthermore, the current prototypical implementation of ProcessExplorer shows all the detailed information about the subset and insight recommendations at the same time. It is therefore interesting to further investigate other user interface concepts.

7.6.2 Usefulness

The evaluation of ProcessExplorer was limited to the usefulness aspect of the system. Although the results have shown significant usefulness of the underlying concepts, it is still unclear how the recommendations specifically improve the quality of the analysis. We also did not investigate potential negative aspects of the approach, such as the influence of recommendations

on analysts to only inspect the suggested process characteristics. Furthermore, the evaluation did not investigate the actual time improvement during the analysis because our system was only implemented as a prototype with limited capabilities. Integrating the presented approach into an enterprise process mining tool is necessary for a fair comparison.

7.6.3 Runtime Performance

The runtime performance of finding interesting insight recommendations highly depends on the PPIs considered as well as on the number of investigated cases. During the user studies, the evaluation of the statistical significance tests took the majority of the runtime; this part needs further improvement and optimization. Sampling strategies, as well as data structure efficiency, are promising topics of investigation.

7.7 Conclusion

This chapter introduced ProcessExplorer, an interactive visual recommendation system for process mining which automatically provides the analyst with subset and insight recommendations ranked by interestingness. In summary, the main contributions of this chapter are:

1. A novel approach to automatically compute recommendations that guide process analysts during the analysis of large event logs. Specifically, this chapter introduced several innovative techniques:
    a) A method to automatically obtain subset recommendations from event logs using multi-perspective trace clustering, considering the control flow and data perspective of cases.
    b) A mechanism, based on statistical significance testing, that identifies the most deviating PPIs that are relevant and insightful for the analyst to explore.
    c) A diversifying top-k ranking approach for subset and insight recommendations to generate a ranked list of diversifying recommendations.
2. An interactive visual exploration system that enhances analytic support to quickly explore large and complex event logs.

ProcessExplorer enhances existing process mining tools by suggesting subset and insight recommendations to support the analyst through the exploration of large and complex event logs. The basis for ProcessExplorer is the process knowledge artifact framework, introduced in Chapter 3 and illustrated in Figure 7.13. Subset recommendations are obtained by the multi-perspective trace clustering that was introduced in Chapter 6 and allows

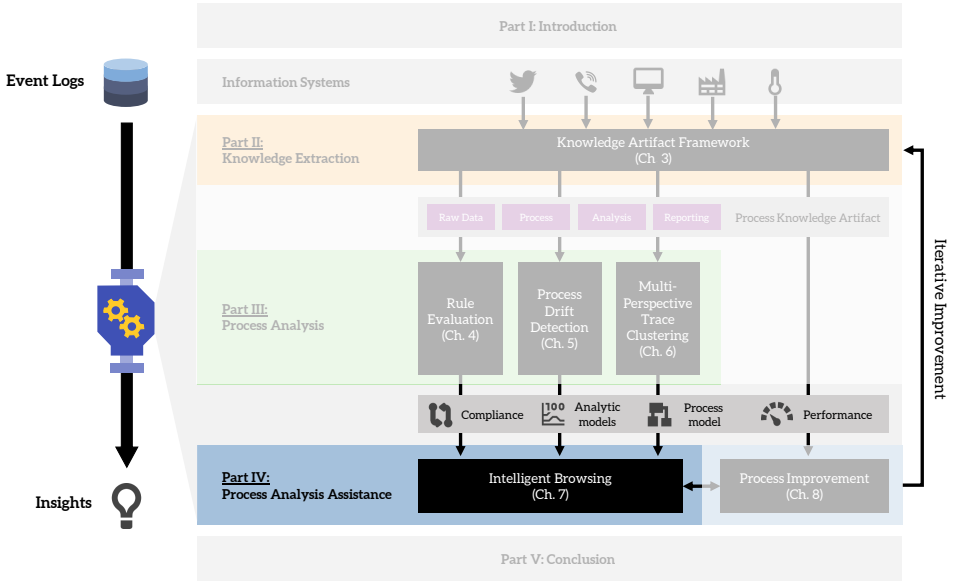


Figure 7.13: Overview of the chapter and contributions.

analysts to quickly scan through the most interesting process behavior patterns. Insight recommendations computed from a predefined set of PPIs are filtered by relevance using statistical significance analysis and sorted by interestingness using the effect size. As a result, ProcessExplorer provides interactive exploration of event logs combined with automatically generated recommendations to guide analysts towards interesting findings. In the following chapter, we introduce a process improvement algorithm that analysts can configure based on the insights obtained by ProcessExplorer.

8

Process Improvement

The methods presented in previous chapters analyze business processes by inspecting recorded event logs, but they do not necessarily provide process improvements. This chapter introduces a process model improvement algorithm using motif-based graph adaptation. The underlying assumption is that there exists a correlation between fundamental metrics of a process map and the relative occurrence of motifs. These correlations were observed in several studies in network theory. In this chapter, the same idea of adapting network topologies is transferred to the context of business processes. The process map is automatically adapted according to a given improvement goal using a motif target signature while retaining important process constraints. The algorithm provides suggestions for redesigning the business process by applying the Local Motif-based Adaptation (LoMbA) algorithm. It automatically adapts the process map by adding, removing, or moving transitions to approximate a target signature.

This chapter is organized as follows: First, in Section 8.1 a motivating introduction to process model improvement via motif-based graph adaptation is given. Next, related work in the field of process optimization is introduced in Section 8.2. Section 8.3 presents a method to obtain motif target signatures for specific process improvement goals. Then, Section 8.4 presents the motif-based graph adaptation algorithm that improves process maps without affecting the process goal. In Section 8.5 the results of the experimental evaluation are presented. The chapter concludes with the limitations of the algorithm, and future work (Section 8.6).

> **Publication:** This chapter is based on the following publication:
>
> Alexander Seeliger, Michael Stein, and Max Mühlhäuser. "Can We Find Better Process Models? Process Model Improvement Using Motif-Based Graph Adaptation." In: *Business Process Management Workshops*. Cham: Springer International Publishing, 2018, pp. 230–242. DOI: 10.1007/978-3-319-74030-0_17.
>
> **Contribution Statement:** I led the idea generation, implementation of the prototypes and performed the data evaluation. The student *Alex Fedjakin* implemented the prototype for the evaluation. *Michael Stein*

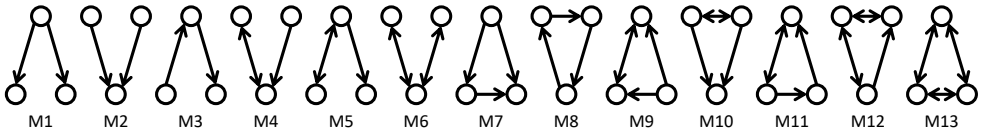


Figure 8.1: Overview of all directed motifs with 3 nodes

> and *Max Mühlhäuser* supported the conceptual design and contributed to the writing process.

8.1 Introduction and Motivation

The success of organizations highly depends on the efficiency, effectiveness, and reliability of business processes. Many organizations invest a lot of effort into the analysis and optimization of business processes to be successful in the competitive market. The previous chapter in Part iii investigated process analysis techniques that obtain valuable insights into how business processes are actually executed. A drawback of these methods is that they do not necessarily provide potential improvement advice [ALS16]. The automatic improvement of process models is a highly complex and challenging problem because systems must provide better alternatives for existing process models only using extracted knowledge from event logs [Aal13].

Two main goals must be fulfilled to improve business process models (see Figure 8.2):

IMPROVEMENT GOAL. On the one hand, the desired and intended improvement goal must be achieved. For instance, from an organization point of view it may be desired to improve the overall throughput time of the procurement handling process to save money.

PROCESS GOAL. On the other hand, the process must still be executable and deliver the desired outcome after the suggested improvements are applied to the process model (see Section 2.2.3). For example, the procurement handling process has the goal that ordered goods are received and paid.

Only if both goals are fulfilled, suggested process improvements are valuable for organizations.

This chapter introduces a process model improvement approach that suggests adjustment recommendations for existing process models based on defined improvement goals. The proposed approach is inspired by the systematic adaptation of communication network topologies based on motifs [Kru+10; Kru+11; Ste+17]. A motif is a small graph pattern that is the core building block of complex graphs [Mil+02] (see Figure 8.1). The frequency distribution

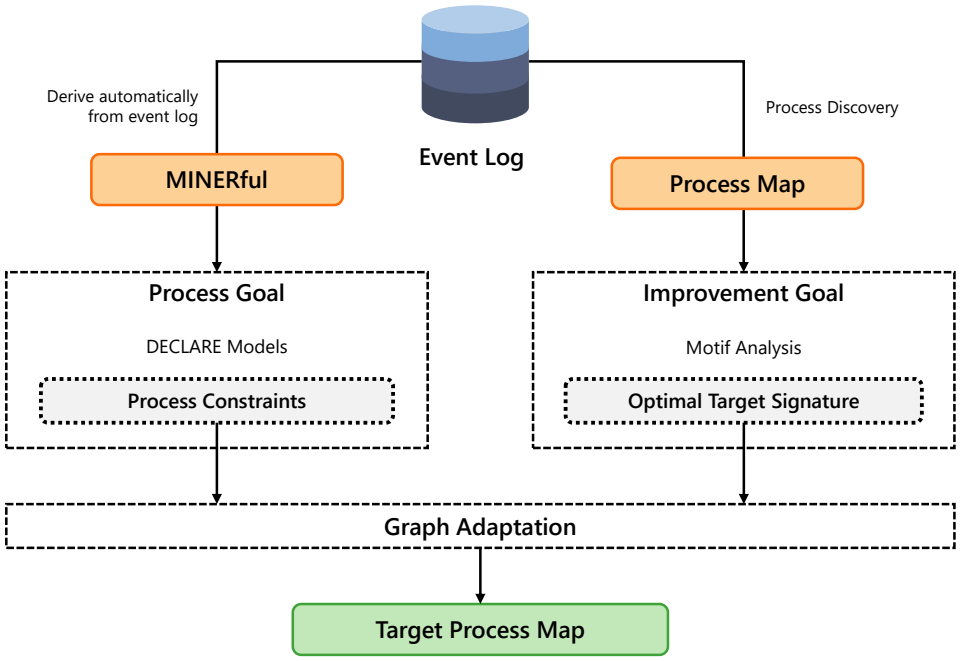


Figure 8.2: Overview of the business process motif-based graph adaptation approach [SSM18].

of motifs, also called motif signature, has been investigated in several studies [Mil+02; SS10] for different types of graphs. These studies consistently observed a strong correlation between the motif signature and important structural metrics of the graph. This observation is utilized in communication network approaches to adapt their topology [Kru+10; Kru+11; Ste+17] as follows: First, a target signature is computed from a set of network topologies that perform well for the investigated metrics. Second, a badly performing network topology is adapted such that the modified topology approximates the given target signature. We transfer the same idea to process model optimization to adapt business process models.

The input of the process model optimization approach is a process map, an event log, and the improvement goal as a motif target signature. A motif signature of a process map represents the core structure of the process. Consequently, it is also an indicator of process simplicity and variability. Similar to the optimization of network topologies, an appropriate motif target signature is defined that is used to adapt non-optimized process models. A process map can be modified by adding or removing transitions between activities. If we modify the process map such that its structure approximates the target signature, the resulting process map has the same core structural properties as the process maps from which the target signature were obtained. The LoMbA algorithm [Ste+17] performs the systematic adaptation of the process map by modifying it according to the target signature. Furthermore, LoMbA ensures that after the adaptation the process goal is still fulfilled

using constraints obtained from an event log or manually from process experts.

8.2 Related Work

Business process optimization is largely a manual task for which a wide range of best practices and guidelines have been established [VTM08; KR13]. Workshops with external consultants are often conducted to brainstorm about process improvements [MRO09]. An overview and quantitative evaluation of process redesign practices are compared in [RL05; Van+15]. As a result, a set of process redesign techniques is proposed that is structured as a pattern catalog to redesign processes. During the design phase of a process model, a wide range of modeling tools exist that integrate such catalogs helping organizations to apply best practices directly to the designed model [GK02]. Only little work is published from a technical and improvement perspective in the context of process mining [VTM08; ALS16]. In the following, we present related work on (semi-)automatic process model improvement techniques.

An optimization approach that combines reference models and process mining is introduced by Gerke et al. [GT09]. The basic idea is to use the IT Infrastructure Library (ITIL) – which consists of a collection of best practices for designing, controlling and improving IT services – and to compare them with the actual process model. Conformance checking identifies the deviations between the reference and the obtained process model to reveal potential improvements.

Niedermann et al. [NRM10b; Nie15] introduce a semi-automatic process optimization platform supporting the process design, execution, and analysis based on historic event logs. Best practices combined with analytical components are stored in a pattern catalog [NRM10a]. Each pattern consists of a formal detection description and an application part, describing the transformation logic to optimize the desired process property. The system automatically matches appropriate patterns and lets the analyst decide which pattern to apply.

Another approach for optimizing the performance of process models is introduced by Yilmaz et al. [YK15], who use cross-organizational process mining to provide improvement suggestions. Process Performance Indicators (PPIs) and process models are computed from different event logs originating from organizations. The differences among the organizations are identified and analyzed to suggest improvements that can be applied to the other organization.

8.3 Finding Appropriate Target Signature

In this section, we discuss how a suitable target motif signature can be obtained to adapt a process map according to a specific improvement goal. The target motif signature is a projection of the desired structural characteristics of the process map, interpreted as a graph. Following the LoMbA approach introduced, it fundamentally influences which modifications are applied to a process map and how the resulting process map should look.

First, we introduce the definitions of a motif and a motif signature, before investigating the motif signatures of real-life process maps:

Definition 8.1 (Motifs [SSM18])**.** A motif is a small subgraph pattern; most practically relevant motifs contain 3 or 4 nodes. Let M be the finite set of motifs of the same number of nodes.

Figure 8.1 shows all 13 different directed motifs with 3 unlabeled nodes. The frequency of how often the different motifs are observed in a process map is called motif signature, and defined as follows:

Definition 8.2 (Motif Signature [SSM18])**.** Let P be the process map, according to Definition 2.9. The frequency of M in P is the motif signature $s(P)$ which is a real-valued vector of relative frequencies of motif M_i with i being the index of the motif.

To get a better understanding of motif signatures for process maps, we investigate five different event logs and examine the signatures for three-node directed motifs. For each event log, a process model is discovered using the Flexible Heuristics Miner (FHM) (see Section 2.4.2) with the standard and "noise-free" setting. In total, ten process models were investigated.

The results of the investigations for the standard heuristic indicate that motifs M1 to M3 are the most dominant motifs in process maps. The motifs M4 to M13 are less present. This distribution of the motifs is not surprising because process maps consist of start and end activities, and transitions between activities link them together. The high frequency of motifs M1 and M2 can be explained by the fact that both motifs refer to the split and join constructs of AND and OR gates. Motif M3 refers to a simple forward transition. Motifs M4 to M6 can be observed when an activity of the process map is executed again after another activity was executed. If the process map contains cycles, motifs M7 and M8 may occur. The motifs M9 to M13 are combinations of other motifs and are not likely to occur frequently in a process map. The detailed distribution of motifs is depicted in Figure 8.3.

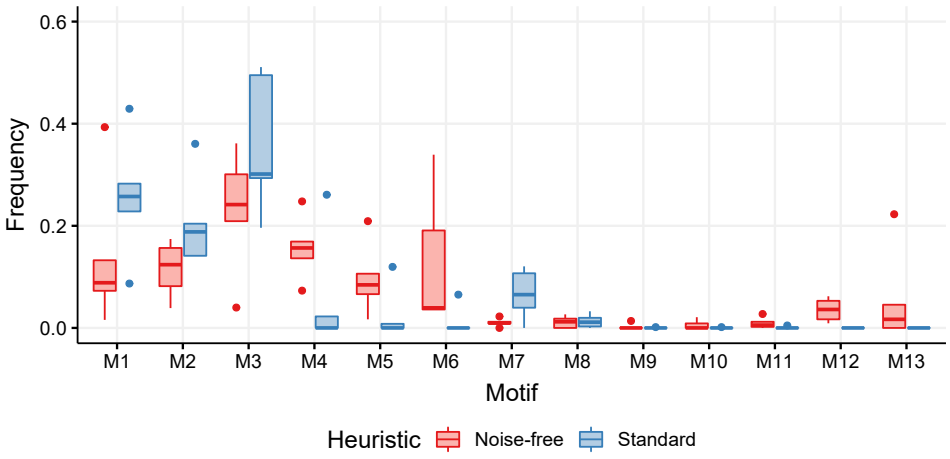


Figure 8.3: 3 node motif frequency over the investigated process models with two different heuristic miner settings.

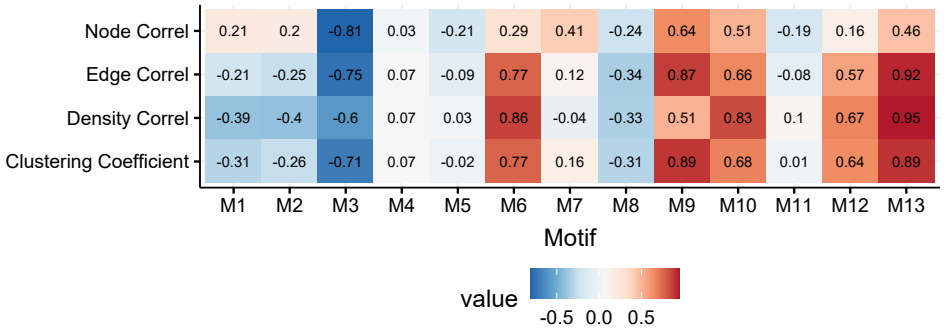


Figure 8.4: Pearson correlations between motif and graph property. Correlations > 0.55 are significant with $p < 0.05$; correlations > 0.6 with $p < 0.01$.

The noise-free setting shows a slightly different result. The results indicate a higher occurrence of motifs M4 to M6 and a lower frequency of motifs M1 to M3. This can be explained by the fact that the FHM in the noise-free setting returns process models with less frequent behaviors. The comparison between the noise-free and the standard setting is depicted in Figure 8.3.

The investigations also show that there exist correlations between motifs and certain process map characteristics. The number of edges, the number of activities, the graph density, and the clustering coefficient were used as characteristic measures. Results of the correlation analysis indicate the following correlations:

- M3 has a significant[1] negative correlation with the number of edges, the number of nodes, the graph density, and the clustering coefficient.

1 Analysis of variance: $p < 0.01$ for correlations > 0.74, and $p < 0.05$ for correlations > 0.55

- M6, M9, M10, M12, and M13 have a significant positive correlation with the number of edges.
- M6, M10, M12, and M13 have a significant positive correlation with the graph density.
- M6, M9, M10, M12, and M13 have a significant positive correlation with the clustering coefficient.

The detailed results of the correlation analysis are depicted in Figure 8.4.

Based on the results of the investigations, target motif signatures can be obtained. One option to obtain a target motif signature is to take the results of the analyzed event logs and make use of the correlations identified. For example, if the reduction of complexity of a process is chosen as the improvement goal, the frequency of the motifs with a positive correlation to the graph density should be decreased. Another option to obtain a target motif signature is to take the motif signature of an existing process map that is already less complex.

In the remainder of this chapter we are interested in improving a process map with respect to model simplicity measure (see Section 2.4.3.4), i. e., the process map should be simplified regarding the number of edges, the graph density, and the cluster coefficient. However, note that the introduced approach is not limited to this specific process improvement goal.

8.4 Motif-based Process Adaptation

This section introduces the process improvement approach. Given a target motif signature and an event log, the process map is adapted such that the resulting process map approximates the target signature while retaining important process constraints. The algorithm is composed of two parts:

1. The LoMbA algorithm adapts the process map and provides process improvement suggestions that fulfill the process constraints given to the algorithm (Section 8.4.1).
2. We propose the use of DECLARE models that describe the process constraints required to successfully fulfill the process goal of the process. DECLARE models can be either defined manually or obtained automatically from an event log (Section 8.4.2).

In the following, each step is described in detail.

8.4.1 Process Map Adaptation

This section describes the motif-based graph adaptation algorithm. The algorithm adapts the process map such that it approximates the target motif

signature and fulfills a specified set of process constraints. Let $P = (N^*, E^*)$ be the process map and t be the target signature. Furthermore, let f be the process constraint function that checks if the process map fulfills the process constraints. The process constraint function is defined as follows:

Definition 8.3 (Process Constraint Function)**.** Let $f : P \rightarrow \{true, false\}$ be a function that receives a process map P and returns a boolean value. f returns *true* if the process map P fulfills all specified process constraints, otherwise *false*.

The motif-based graph adaptation algorithm evaluates the process constraint function (for details see Section 8.4.2) to check whether the application of a set of possible process map modifications still leads to a valid execution of the process. Specifically, the process constraint function validates the fulfillment of the process goal by the given process map.

For the actual adaptation of the process map, a modified version of the LoMbA algorithm [Ste+17] is used. It is an iterative algorithm that periodically traverses all activities $n \in N^*$ of the process map. The algorithm consists of two steps, executed in a loop until the target signature distance is smaller than a defined threshold or a maximum number of iterations is reached:

1. *Iterator Order.* The algorithm searches for a reasonable set of possible graph operations that can be applied to the process map.
2. *Operation Selection.* It selects a graph operation that fulfills the process constraint function and checks if the operation adapts P towards the target signature.

8.4.1.1 Iterator Order

In the first step, LoMbA searches for possible graph modification operations that can be applied to the process map. The algorithm considers three graph adaptation operations: the *Remove-edge*, *Add-edge*, and *Move-edge* operations. Even a small process map leads to a large number of possible graph adaptation operations. Therefore, LoMbA only considers a subset of all graph operations. The algorithm uses the *edge operator indicator* (EOI) heuristic to reduce the operator search space. This heuristic computes which graph operations are most beneficial to approximate P towards the target signature. It is defined as follows:

$$EOI = \frac{\sum(t_i - s_i(P)) \cdot |E(M_i)|}{l} \tag{8.1}$$

where t_i is the relative frequency of motif M_i in the target signature, $s_i(P)$ the relative frequency of motif M_i in the process map P, $|E(M_i)|$ the number of edges in motif M_i, and l the number of inspected motifs. The EOI is used

to compare the ratio of edges in the target signature with the ratio of edges in the current process map signature:

- If $EOI > 0$ then the target signature contains more edges than the current process map signature, indicating that the *Add-edge* operator is likely more appropriate to approximate the target signature.
- If $EOI < 0$ then the target signature contains fewer edges than the current process map signature, indicating that the *Remove-edge* operator is likely more appropriate.
- If $|EOI| \leq eoi\_threshold$, then the ratio of edges between the target and the current signature is too small to promise significant progress in case of$Add - edge$ or $Remove - edge$. Consequently, the $Move - edge$ operator is likely to be more appropriate.

Based on the EOI value, a set of ranked operations is considered for the adaptation step.

8.4.1.2 Operation Selection

In the second step, LoMbA selects an appropriate graph operation of the generated operator iterator for $n \in N^*$ and generates a candidate process map P' by applying the selected operation. The candidate process map is checked to determine whether the process constraint function returns true or not. If P' does not fulfill all process constraints, the selected operation is discarded. Otherwise, the algorithm calculates the motif signature distance between $s(P')$ and t. The motif signature distance is defined as follows:

Definition 8.4 (Motif Signature Distance)**.** The $distance(x, y)$ between two motif signatures x and y is the Euclidean distance between x and y.

After the algorithm generated the candidate process map P', it checks if the signature distance of the candidate process map is closer to the target signature t than the current process map. The modification operation is accepted and P' is the new current process map. If the operation was accepted, the current operator iterator may contain further appropriate modifications. In this case, the algorithm continues to apply as many other operations of the iterator as it already applied to P. However, if no operation was found that fulfills the process constraints, a different operator iterator is generated by expanding the search space. As opposed to the original LoMbA algorithm, the algorithm operates on the entire graphs, not on subgraphs, because process maps contain fewer nodes and edges than network topologies.

The algorithm continues to adapt the process map until the maximum number of iterations is reached. The experimental results show that a maximum number of five iterations is sufficient to appropriately approximate the tar-

Algorithm 6: Motif-based Process Graph Adaptation Algorithm

input: $P = (N^*, E^*), t$, eoiThreshold

its ← {`Remove-edge` (P, n), `Add-edge` (P, n), `Move-edge` (P, n)}
EOI ← `GetEdgeIndicator` (P, t)
itOrder ← `GetIteratorOrder` (its, EOI, eoiThreshold)
for *it: itOrder* **do**
    foundValid ← false, maxSteps ← ∞, doneSteps ← 0
    currentDistance ← `Distance` (s(P), t)
    while *op ← it.next() and* doneSteps < maxSteps **do**
        doneSteps ← doneSteps + 1
        P' ← `CreateCandidate` *(P, op)*
        if *f(P′)* **then**
            foundValid ← true
            if `Distance` *(s(P′), t)* < currentDistance **then**
                $P \leftarrow P'$
                if maxSteps = ∞ **then**
                    maxSteps ← doneSteps
                end
                doneSteps ← 0
                currentDistance ← `Distance` (s(P), t)
            end
        end
    end
    if foundValid **then** break;
end

DECLARE rule	LTL Expression
responded existence(A, B)	$\diamond A \Rightarrow \diamond B$
co-existence(A, B)	$\diamond A \Leftrightarrow \diamond B$
response(A, B)	$\square(A \Rightarrow \diamond B)$
precedence(A, B)	$(\neg B \sqcup A) \vee \square(\neg B)$
succession(A, B)	$response(A, B) \wedge precedence(A, B)$
alternate response(A, B)	$\square(A \Rightarrow \bigcirc(\neg A \sqcup B))$
alternate precedence(A, B)	$precedence(A, B) \wedge \square(B \Rightarrow \bigcirc(precedence(A, B)))$
alternate succession(A, B)	$alternate\ response(A, B) \wedge alternate\ precedence(A, B)$
chain response(A, B)	$\square(A \Rightarrow \bigcirc B)$
chain precedence(A, B)	$\square(\bigcirc B \Rightarrow A)$
chain succession(A, B)	$\square(A \Leftrightarrow \bigcirc B)$

Table 8.1: Semantics for the DECLARE constraints [Bur+12].

get motif signature. The entire definition of the algorithm is illustrated in Algorithm 6.

8.4.2 Specification of Process Constraints

This section describes how process constraints can be described to preserve the process goal and the feasibility of the process. For this purpose, DECLARE [PSA07] models are used to specify the process constraints. DECLARE is an LTL-based declarative process modeling language that uses a graphical representation with formal semantics to model the behavior of a process via instantiation of templates [Bur+12]. The templates are listed with their corresponding LTL expressions in Table 8.1.

The following symbols are used to express the semantics of the temporal operators:

Operator	Symbol	Explanation
Next	$\bigcirc\phi$	ϕ has to hold the next state.
Finally	$\diamond\phi$	ϕ eventually has to hold.
Globally	$\square\phi$	ϕ has to hold on the entire subsequent path.
Until	$\psi \sqcup \phi$	ψ has to hold at least until ϕ becomes true, which must hold at the current or a future position.

Table 8.2: Symbols for the LTL expressions.

Each template specifies a constraint that needs to be fulfilled by the process model. For instance, the *response* template defines that each time a specific activity is executed it must be eventually followed by another specific activity.

With the templates, a wide range of different process constraints can be specified. These constraints are evaluated in the adaptation step to check whether a proposed adaptation of a process map is still feasible. Each time the adaptation algorithm applies a possible graph operation, it executes the process constraint function to check the predefined DECLARE models.

There are two approaches to obtain DECLARE models for describing process constraints that must be fulfilled for suggested process improvements. The first approach is to manually specify DECLARE models using the graphical representation editor [Bur+12]. However, specifying DECLARE models that fully describe the process constraints may be difficult when processes get more sophisticated. The second approach is to obtain DECLARE models from event logs automatically. This is particularly interesting because DECLARE models obtained from real process executions better reflect the actual process. MINERful [Di +15] is such a declarative process discovery tool that obtains a DECLARE model from an event log. The tool allows the analyst to control the quality of the DECLARE model by specifying a confidence threshold which is the product of support and the relative frequency of traces in the event log containing the specific DECLARE rule. The model obtained is then utilized for the adaptation algorithm as the input for the process constraint function to generate valid process maps.

8.5 Evaluation

This section presents the evaluation results of the motif-based graph adaptation algorithm for improving process maps.

8.5.1 Experiment Setup

In the evaluation, five real-life event logs (see Table 8.3) were used to show how the proposed approach adapts the process maps to achieve the improvement goal. For each event log, two process maps were generated with the standard and the noise-free setting of the FHM. The process constraints for the event logs were obtained by means of the MINERful tools with a confidence threshold of 0.95. Table 8.3 shows the number of DECLARE rules that were obtained by the tool.

The improvement goal for all event logs is to increase process model simplicity. Based on the observations in Section 8.3 three different target signatures are defined:

1. $t_1 = \{0.33, 0.33, 0.33, 0, 0, ...\}$
2. $t_2 = \{0.2, 0.2, 0.6, 0, 0, ...\}$
3. $t_3 = \{0.228, 0.228, 0.347, 0, 0, 0, 0, 0.19, 0, 0, ...\}$

Event log	Instances	Variants	Events	Events/Case	Constraints
BPIC '12	13087	7179	36	20.04	214
Large	651709	30460	35	5.95	122
Small	873	101	45	7.77	26
Midsize	90536	1630	30	9.06	56
Environmental	1434	381	27	5.98	49

Table 8.3: Characteristics of the used real-life event logs and the number of extracted process constraints.

The motif-based graph adaptation algorithm is executed until the distance between the actual and the target signature is less than 0.1, which was achieved in a maximum of five iterations.

8.5.2 Results

This section presents the results of the three evaluation objectives: the improvement goal performance, the distance changes to the target signature per iteration, and the number of applied operations per iteration.

8.5.2.1 Improvement Goal Performance

The main objective of this evaluation is to compare the three different target signatures and investigate which signature achieves the improvement goal best. For all event logs, the initial number of edges, the graph density, and the clustering coefficient are measured. After executing the graph adaptation algorithm using the three different target signatures, the resulting measurements are obtained. The results of the improvement performance are illustrated in Figure 8.5.

First, we investigate the process models obtained using the standard setting for the FHM. The results indicate that only the target signatures t_1 and t_2 decreased the number of edges, the graph density, and the clustering coefficient, which was the improvement goal to achieve. Target signature t_3 increased the clustering coefficient and only removed three edges from the *Environmental* event log. Although t_3 was derived from the correlation analysis conducted in Section 8.3, this target signature did not perform well for our improvement goal. The detailed results are illustrated in Table 8.4.

The process models obtained with the noise-free setting of the FHM contain more edges, have a higher graph density, and a higher clustering coefficient. After the motif-based graph adaptation, the target signatures t_1 and t_2 produced process models with fewer edges and a lower clustering coefficient. Again, for target signature t_3 the number of edges and the clustering coeffi-

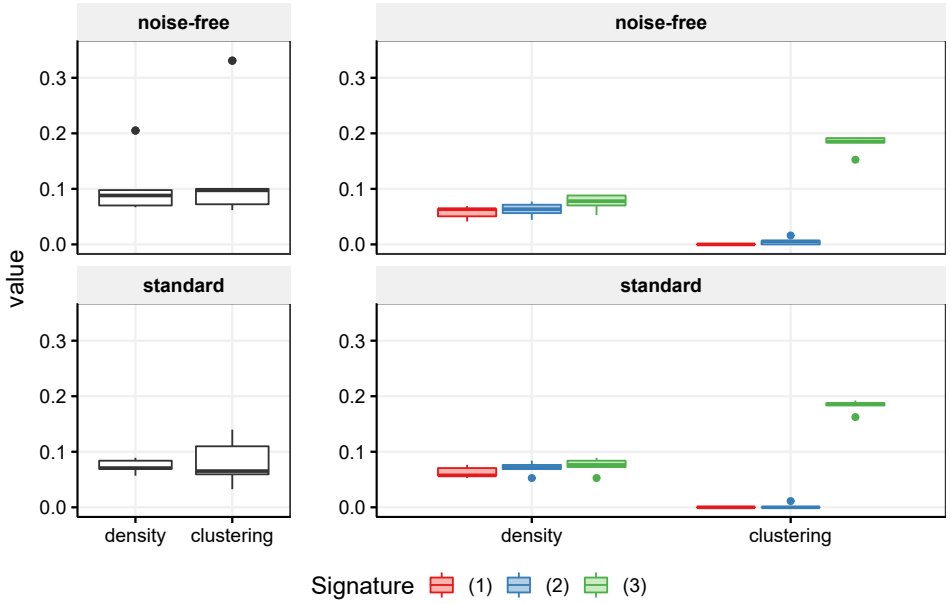


Figure 8.5: Cluster coefficient and density values before and after the optimization grouped for the three different target signatures.

cient is not significantly improved, except for the event log *Large*. The detailed results are shown in Table 8.5.

Based on the results of both experiments, we come to the conclusion that the achievement of the improvement goal depends strongly on the definition of the target motif signature. Although the three target motif signatures obtained from the correlation analysis vary only slightly, these small differences have a high influence on the resulting process models. We showed that the desired improvement goal was achieved for two manually defined target motif signatures. For both target motif signatures, the distribution was chosen so that three motifs make the greatest contribution. However, our experiments did not show a general rule of thumbs that would suggest how to exactly define the target motif signature.

8.5.2.2 Distance to Target Signature over Iterations

Another evaluation objective is how quickly the motif-based graph adaptation algorithm approximates the target signatures for a given process model under the process constraints. Figure 8.6 shows the distance to the target signature across all event logs for each found. The results show that for almost all process models, the distance to the target signature converges towards zero within the five iterations. The target signature t_1 is reached already within four iterations for all evaluated process models. However, the target signature t_3 could only be approximated, and only for the small process model, the distance to the target signature is zero. In all cases, the largest jump towards

		BPIC'12	Large	Small	Midsize	Environ.
Initial Edges		39	106	29	73	40
Initial Density		0.071	0.089	0.069	0.084	0.057
Initial Cluster Coefficient		0.059	0.120	0.065	0.110	0.033
t_1	Δ Edges	-1	-20	0	0	-3
	Δ Cluster Coefficient	-0.059	-0.140	-0.065	-0.110	-0.021
t_2	Δ Edges	0	-38	0	-23	-3
	Δ Cluster Coefficient	-0.059	-0.140	-0.065	-0.110	-0.033
t_3	Δ Edges	1	0	0	0	-3
	Δ Cluster Coefficient	0.133	0.023	0.122	0.076	0.151

Table 8.4: Difference of the number of edges, graph density, and clustering coefficient for the process models obtained using the standard setting of the FHM before and after motif-based graph adaptation.

		BPIC'12	Large	Small	Midsize	Environ.
Initial Edges		54	244	37	61	47
Initial Density		0,098	0,205	0,088	0,070	0,067
Initial Cluster Coefficient		0,100	0,331	0,072	0,062	0,097
t_1	Δ Edges	-19	-152	-7	-12	-16
	Δ Cluster Coefficient	-0.100	-0.325	-0.056	-0.056	-0.097
t_2	Δ Edges	-19	-168	-8	-17	-18
	Δ Cluster Coefficient	-0.100	-0.331	-0.072	-0.062	-0.097
t_3	Δ Edges	-11	-138	0	0	-10
	Δ Cluster Coefficient	0.092	-0.178	0.113	0.130	0.086

Table 8.5: Difference of the number of edges, graph density, and clustering coefficient for the process models obtained using the noise-free setting of the FHM before and after motif-based graph adaptation.

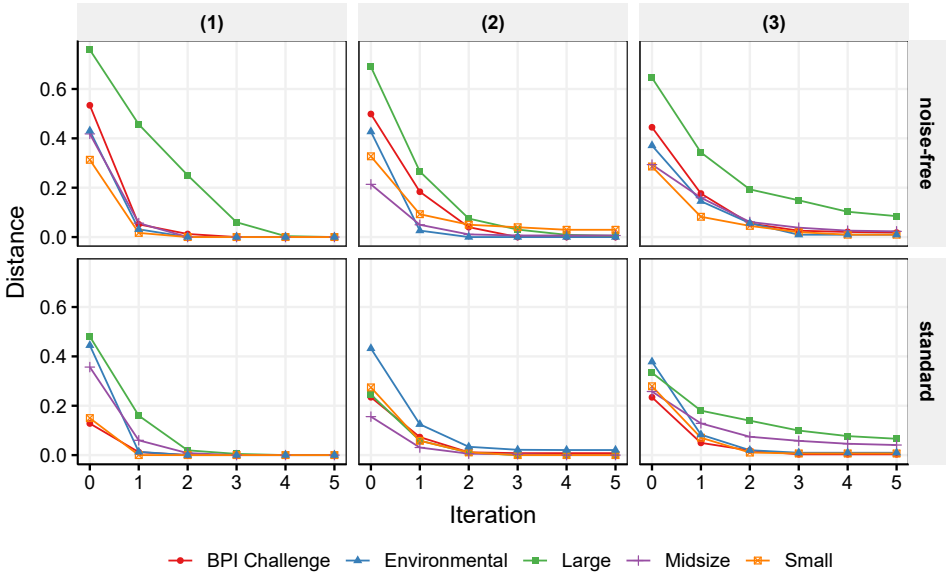


Figure 8.6: Distance to target signatures over the number of iterations.

the target signature is achieved in the first iteration. Another observation is that the size of the process model influences the approximation speed.

8.5.2.3 Operations

Lastly, we investigated the number of removed and added edge operations for each iteration and target signature. The number of applied operations to the process map decreases with each executed adaptation iteration. Most operations were performed in iteration one which was also observed in Section 8.5.2.2. After iteration 2, for almost all investigated event logs, the number of added and removed edges were the same, suggesting that only the move operator was applied. The detailed results of the applied number of operations per iteration are illustrated in Figure 8.7.

8.6 Discussion and Limitations

This chapter presented a process model optimization algorithm based on motif analysis. However, there are some limitations and research directions for future work.

8.6.1 Improvement Goal

The process model optimization algorithm optimizes a given process model according to a target motif signature. Our experiments showed that describing certain process improvement goals as signatures is possible. However, it is

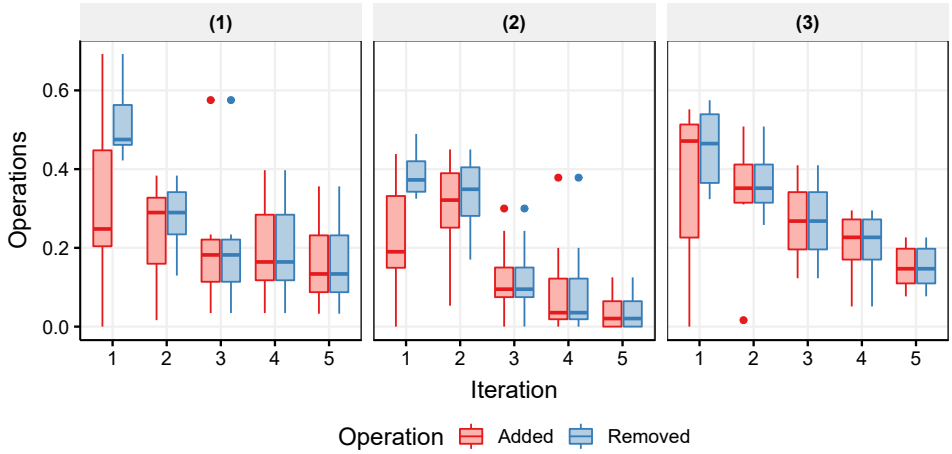


Figure 8.7: Normalized number of edge operations performed for investigated target signatures and iterations.

clear that this is not possible for every process improvement goal. For instance, improving a process model with respect to throughput performance is not possible. Our work is limited to the improvement of process models according to improvement goals that can be specified as a graph metric.

Another limitation of our approach is that it relies on the manual exploration and analysis of correlations between graph metrics and improvement goals. This analysis is necessary to find appropriate target motif signatures. However, finding and analyzing correlations is rather exploratory and, therefore, time-consuming. It may be interesting to look into automating the analysis of correlations between graph metrics and improvement goals to improve the target motif signature definition. Specifying a process improvement goal without the need to specifically design target motif signatures would enhance the applicability of the approach.

A general limitation of process improvement approaches is to measure the quality of the proposed improvement modifications. Specifically, the conducted experimental evaluation did not investigate the resulting process maps after they were adapted. The proposed algorithm ensures that the process constraints defined are fulfilled, but it does not guarantee that the improved process model performs better.

8.6.2 Local Improvements

Motifs are small subgraphs containing only a small number of nodes. Motifs have only a very local perspective on the entire process model and cannot represent larger dependencies. Therefore, our approach is only able to optimize small parts of the process model with the goal that the sum of all smaller optimizations leads to a global optimization.

In the experimental evaluation, only three-node motifs were investigated to specify target motif signatures. A potential future research direction is the use of four or five node motifs which could better reflect the actual structure of a process. However, using larger motifs results in high computation times because counting motifs in graphs is a computationally complex problem [Ste+17].

8.6.3 Process Model

The current algorithm is limited to simple process maps that do not contain complex process semantics such as parallelism. However, more expressive process models are typically used at design time, which requires extensive modifications to the process constraint function. The use of complex process semantics may cause additional mechanisms that check whether branching paths are correctly joined or whether the resulting process model is sound and executable.

8.7 Conclusion

This chapter introduced a motif-based graph adaptation algorithm in the context of business process optimization which automatically provides suggestions for modifying process maps while considering specified process constraints. In summary, the main contributions of this chapter are:

1. A method to retrieve a motif target signature to optimize a process map according to a specific metric.
2. A motif-based graph adaptation algorithm that adapts process maps towards a given motif target signature while retaining important process constraints using DECLARE models.
3. Results from an experimental evaluation investigating the improvement goal performance, the distance to the target signature over iterations, and the number of accepted adaptation operations.

Motif-based graph adaptation in the context of business processes allows the improvement of process maps according to a specific improvement goal without affecting the intended process goal. Based on the insights provided by the algorithms and methods introduced in the previous chapters, analysts can configure the process model improvement algorithm. This chapter introduced the last contribution of this thesis, allowing organizations to implement potential process improvements. It is essential to permanently monitor and evaluate the changes made to the process in an iterative fashion, illustrated in Figure 8.8. In the following chapter, we summarize the contributions of this thesis.

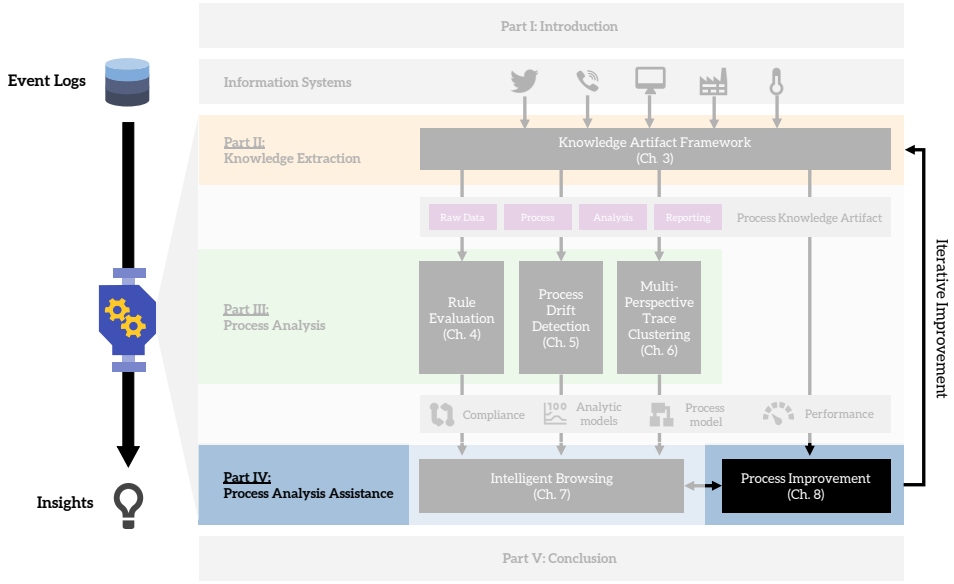


Figure 8.8: Overview of the chapter and contributions.

Part V

Closure

9 Summary and Conclusion

This chapter summarizes the contributions and findings according to the five parts of the thesis. Furthermore, we will give some research directions for future work.

9.1 Summary

This thesis presented several methods and algorithms to provide intelligent computer-assisted process mining for real-life scenarios. In particular, this thesis contributed methods and algorithms for the practical work of analysts within the context of the PM^2 methodology [Eck+15]. Along the workflow of analysts, this thesis addressed common issues found in the state of the art approaches and addressed three main research questions to overcome the challenges related to the practical use of process mining. Specifically, the work is categorized into three main parts.

- **Knowledge Extraction.** Part II of this thesis investigated the consolidation and annotation of heterogeneous data sources to better enable the application of process mining techniques.
- **Process Analysis.** In Part III, this thesis contributed three effective process analysis methods that can be used in real-life scenarios.
- **Process Analysis Assistance.** Part IV of this thesis introduced an intelligent computer-assisted process mining browsing system that guides analysts through the exploration of process models.

In the following, the proposed contributions are summarized and linked to the research questions addressed in this thesis.

Knowledge Extraction

Part II of this thesis deals with the collection, preparation, and data processing of event logs which are the basis for process mining. Although most process mining methods assume the existence of such event logs, it is a non-trivial task to obtain event logs from information systems. Event data is spread throughout the organization and stored in different heterogeneous data sources, not directly usable for process mining. In this context, RQ1 asked

how event logs from different data sources can be collected, consolidated and annotated to better enable their use for process mining.

Chapter 3 introduced the process knowledge artifact framework that significantly simplifies data collection, preparation, and processing for process mining. Our framework integrates the data preparation of raw data of Process-aware Information Systems (PAISs) and the process mining analysis into a combined meta-model. Analysts can model and maintain known and recurring process problems to facilitate process analysis.

Unlike conventional exploratory analysis, where analysts repeatedly perform specific process mining tasks manually, the process knowledge artifact framework enables rapid evaluation for frequently investigated aspects automatically.

Process Analysis

Part III of this thesis deals with the practical use of process mining with real-life event data. Although process mining is a research area of high practical relevance, the growth of data and the increasing complexity of actual processes lead to several issues of current methods. In this thesis, RQ2 asked how real-life event logs can be effectively analyzed using process mining methods. This part of the thesis contributed one targeted and two exploratory process mining analysis techniques.

Chapter 4 addressed RQ2.1, answering how compliance rules can be efficiently evaluated. We contributed a novel compliance checking method that transforms the problem of compliance checking into a graph-reachability problem. Our algorithm is significantly more efficient than comparable rule-based approaches. Furthermore, the method provides counterexamples for violated compliance rules to simplify deviation analysis, enabling detailed diagnosis.

In the area of process discovery, RQ2.2 asked how process drifts can be detected without knowledge about the process. Chapter 5 introduced a novel process drift detection method to detect differences in process executions over time automatically. Each process drift is characterized by annotating the discovered process model with the differences being detected. The results of the evaluation showed an overall higher F1-score and a lower detection delay for the benchmark investigated. We also showed that the extracted process drift characterizations for simple process drift patterns match the ones injected into the benchmark data set.

RQ2.3 asked how different process behaviors on multiple perspectives can be detected to improve process discovery. Chapter 6 contributed a novel multi-perspective trace clustering method. Our algorithm is the first process model-based trace clustering approach that combines control flow and data

process perspective to determine the similarity between cases in an event log. This combination is particularly useful in situations where the process is not only determined by the control flow but also by the attached case attributes. The approach automatically optimizes the resulting process models such that they accurately represent the traces within a cluster. Results of the evaluation showed that our method discriminates the different process behaviors accurately. The approach builds the foundation for process analysis assistance in Chapter 7.

The three process analysis algorithms are specifically designed to obtain additional insights about the process execution that was recorded in an event log. The compliance checking algorithm allows to check compliance rules for very large event logs where other approaches do not yield results. Our process drift detection algorithm identifies different types of process changes without the analysts needing additional knowledge about the process. The multi-perspective trace clustering algorithm significantly improves the state of the art by considering different process perspectives during clustering and by automatically optimizing the result with respect to process model fitness.

Process Analysis Assistance

Part IV aims to provide analysis assistance during the practical use of process mining tools. In particular, RQ3 asked how process analysis assistance can support the analyst's workflow.

In Chapter 7 RQ3.1 is addressed, investigating analysis assistance in process mining tools. We introduced ProcessExplorer, an interactive visual recommendation system for process mining. It extends existing state-of-the-art process mining tools by providing two types of recommendations to support the workflow of analysts analyzing large and real-life event logs. The automatic recommendation engine evaluates a wide range of process performance indicators to point analysts towards interesting findings.

RQ3.2 asked how insights of process mining can result in process model improvements. Therefore, Chapter 8 introduced a motif-based process improvement algorithm to suggest modifications to the process model based on a given improvement goal without affecting the business goal. Our approach aims to propose changes to the process model so that a given improvement goal is achieved while preserving the business objectives of the process.

Unlike conventional process mining tools that do not provide any analysis guidance, ProcessExplorer is the first approach to actively suggest analysis recommendations to better help analysts understand the process. Our process model improvement approach is an attempt to suggest modifications to the process model that may improve the process performance with respect to a given improvement goal.

9.2 Directions for Future Research

This section discusses future research directions with respect to the overall goal of this thesis.

- **Parameter-free algorithms**

  This thesis argued in favor of parameter-free algorithms, and most of the introduced contributions do not require to set any parameter at all. As a result, these approaches can be easily applied in practice without knowing the characteristics of the inspected event data. In practice, a large number of existing approaches are not parameter-free, which makes it unnecessarily challenging to obtain valuable results. Automatic parameter tuning approaches, such as introduced for the process drift detection algorithm and the multi-perspective trace clustering algorithm, are needed for other process mining methods in the future to make process mining more accessible.

- **Establish assistance in other process mining areas**

  In this thesis, we combined multi-perspective trace clustering with Process Performance Indicator (PPI) evaluation to provide exploration assistance for process discovery to the analysts. Other process mining areas, such as conformance checking and enhancement, could equally benefit from the contributions made in this thesis. Similar to the interestingness measure introduced, other measures that evaluate the interestingness of different aspects of the process in conformance checking and enhancement may be needed. The ultimate goal is to combine different process mining areas to offer consistent and useful assistance.

- **Improvement of runtime performance**

  In Part III of this thesis, three process analysis methods that aim to deal with real-life event logs were introduced. We made significant runtime improvements to compliance checking by adapting algorithms from other domains. However, the performance of the process drift detection and the multi-perspective trace clustering algorithm mainly relies on the process discovery algorithm Flexible Heuristics Miner (FHM) [WR11] and trace alignments [Man+15]. Both have been shown in the related work to be highly computationally complex in order to obtain reasonable process models and align traces to compute the model fitness. For improving runtime performance of our work, it may be necessary to investigate approximations with much faster runtime for large event logs.

Part VI

Appendix

A

Business Rule Evaluation using TFA

Detailed Evaluation Results

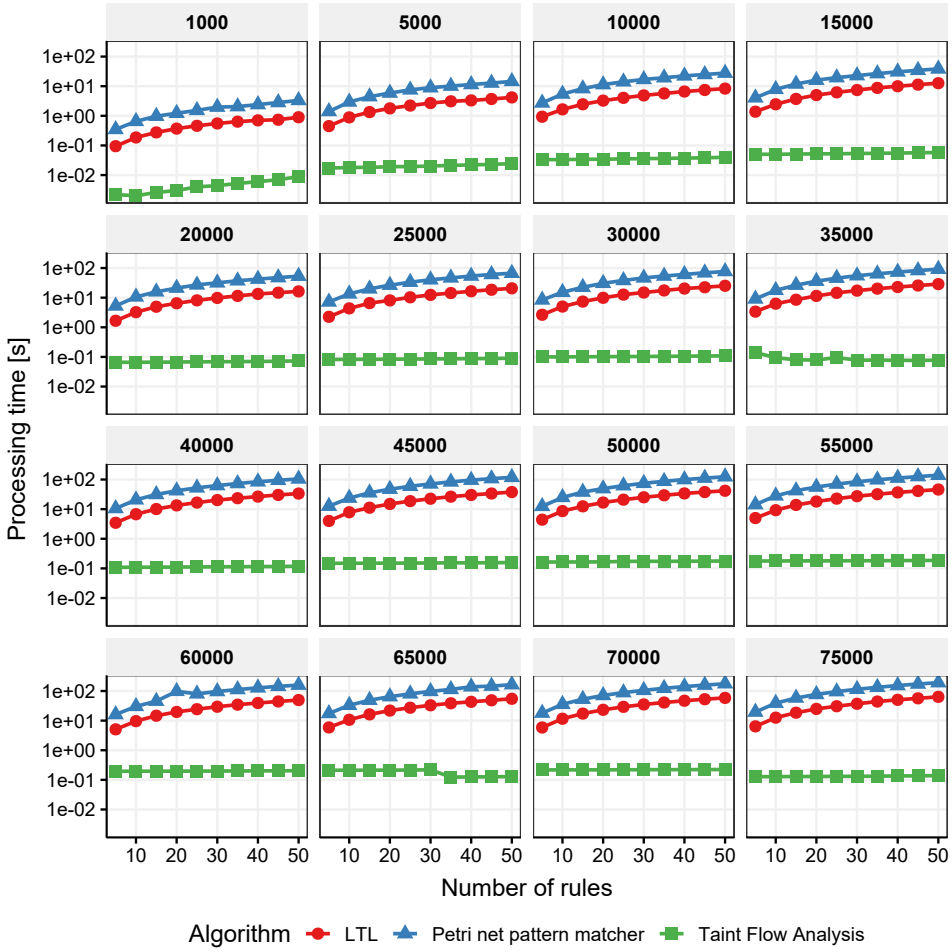


Figure A.1: Processing time in seconds for the synthetic event logs depending on the number of rules in the log (Part 1).

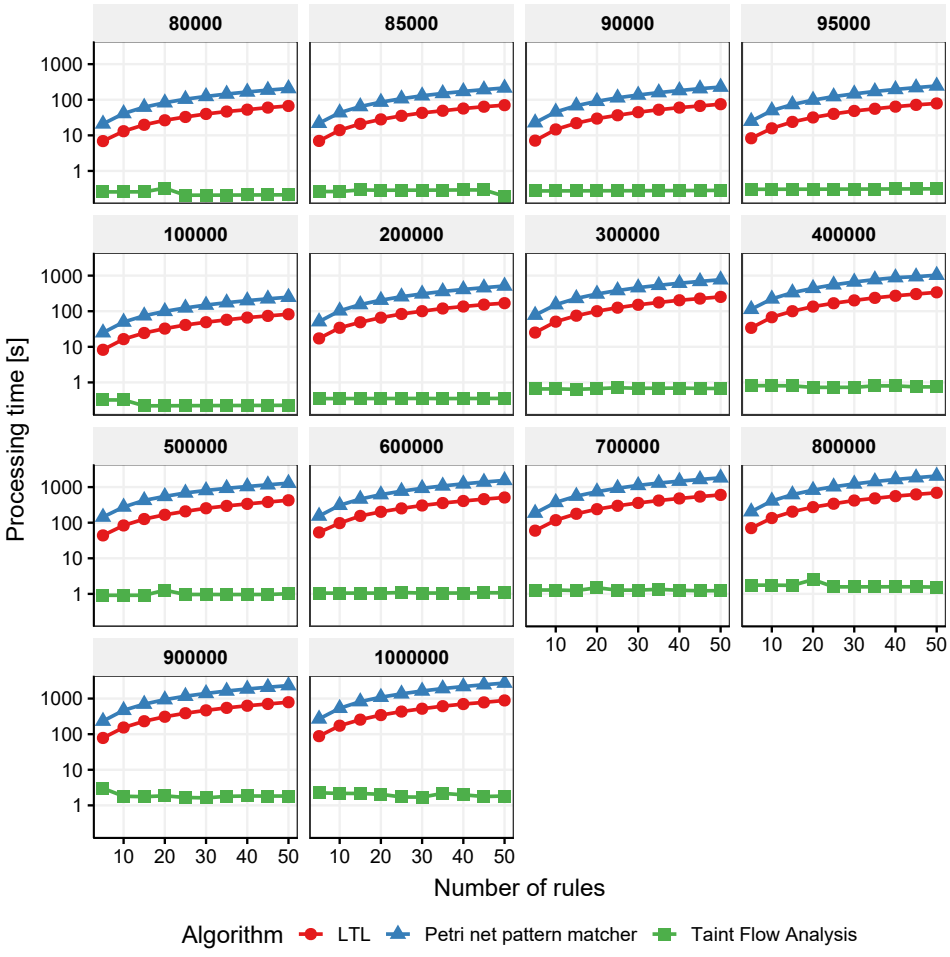


Figure A.2: Processing time in seconds for the synthetic event logs depending on the number of rules in the log (Part 2).

B

Process Drift Detection

Detailed Evaluation Results

Log	F1-Score				Average Distance			
	Bose et al.	Maaradji et al.	Ostovar et al.	Seeliger et al.	Bose et al.	Maaradji et al.	Ostovar et al.	Seeliger et al.
cb	-	0.96	0.72	**0.97**	-	78.38	108.56	**18.94**
cd	**1.00**	-	0.79	0.95	**20**	-	266.78	28.69
cf	0.90	**1.00**	0.81	0.99	36	**32.33**	140.48	34.62
cm	-	0.92	0.81	**0.97**	-	66.24	131.27	**19.24**
cp	0.64	**1.00**	0.89	0.99	36	45.47	224.12	**17.59**
fr	0.44	0.36	0.61	**0.99**	165	65.12	268.18	**19.92**
IOR	0.78	**1.00**	0.92	0.96	38	32.44	200.52	**13.00**
IRO	0.56	**1.00**	0.90	0.95	82	61.11	313.42	**27.22**
lp	0.65	**1.00**	0.89	0.76	**41**	125.58	500.64	48.03
OIR	**1.00**	0.78	0.71	0.73	**20**	111.02	323.19	28.06
ORI	0.78	**1.00**	0.84	0.99	38	41.53	165.70	**14.25**
pl	**1.00**	0.96	0.81	0.96	**20**	37.33	158.17	26.33
pm	0.78	**0.99**	0.88	**0.99**	69	49.56	126.90	**24.78**
re	**1.00**	0.99	0.76	0.90	**17**	48.31	301.61	33.02
RIO	0.56	**0.99**	0.83	0.97	60	48.56	187.88	**20.77**
ROI	**1.00**	**1.00**	0.75	**1.00**	20	37.33	146.20	**7.31**
rp	0.75	**0.97**	0.82	**0.97**	40	45.31	167.46	**12.67**
sw	0.78	**1.00**	0.89	**1.00**	39	46.53	498.05	**29.61**

Table B.1: Detailed results of the experimental evaluation of the process drift detection methods.

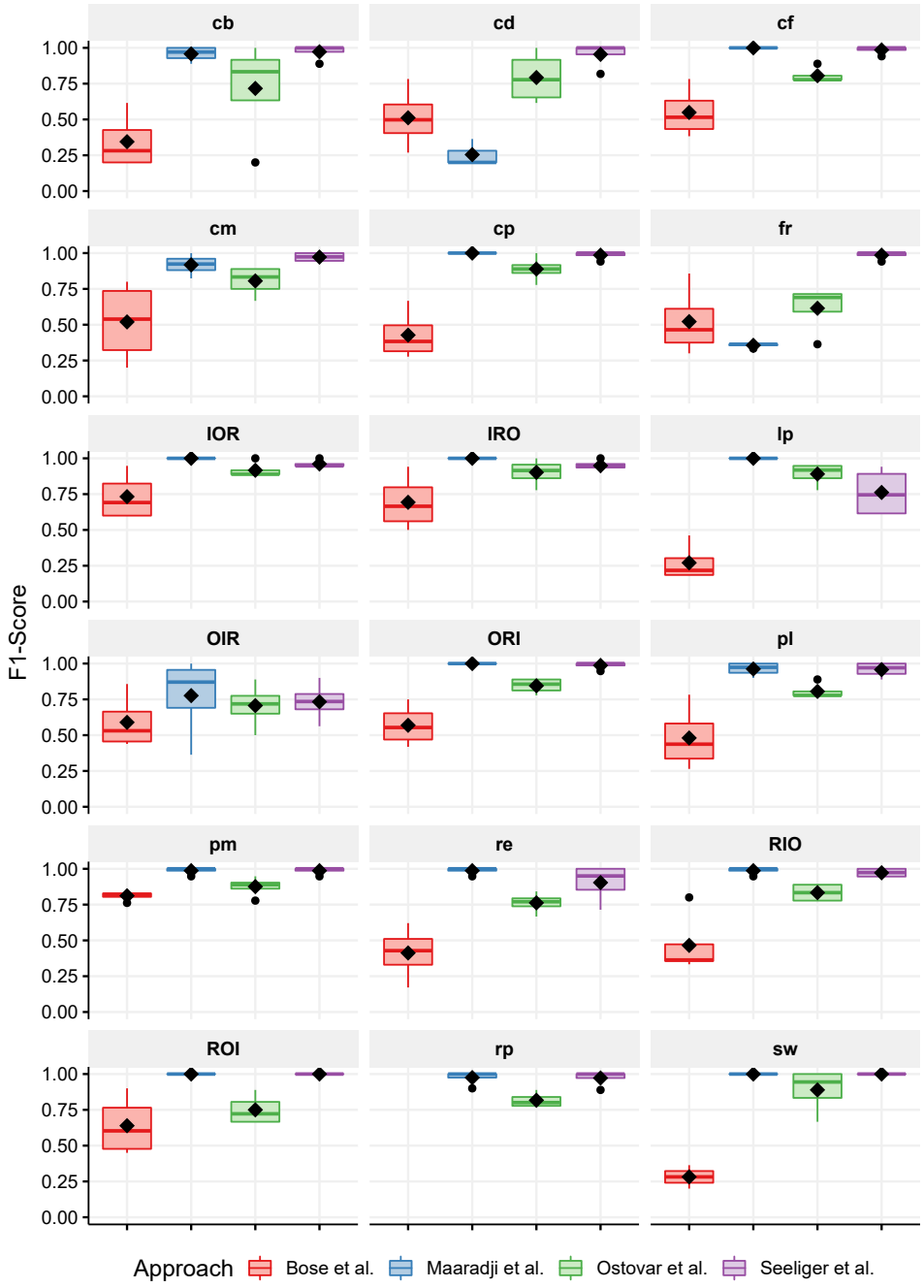


Figure B.1: F1-score for different process drift patterns (higher is better) and compared with the related work. Filled diamond (◆) indicates the mean.

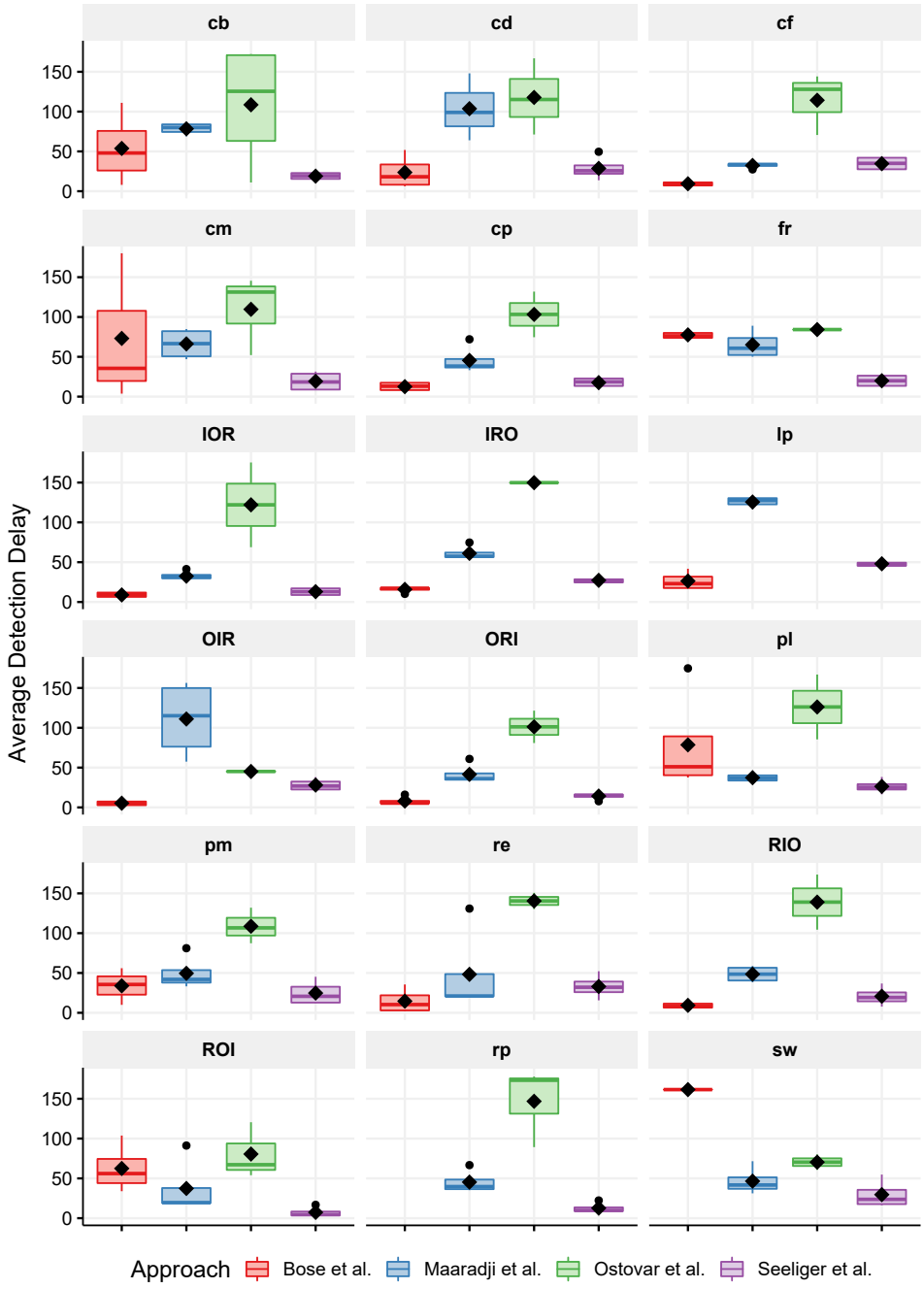


Figure B.2: Average delay for different process drift patterns (lower is better). Filled diamond (◆) indicates the mean.

C

Hybrid Feature Set Trace Clustering

Detailed Evaluation Results

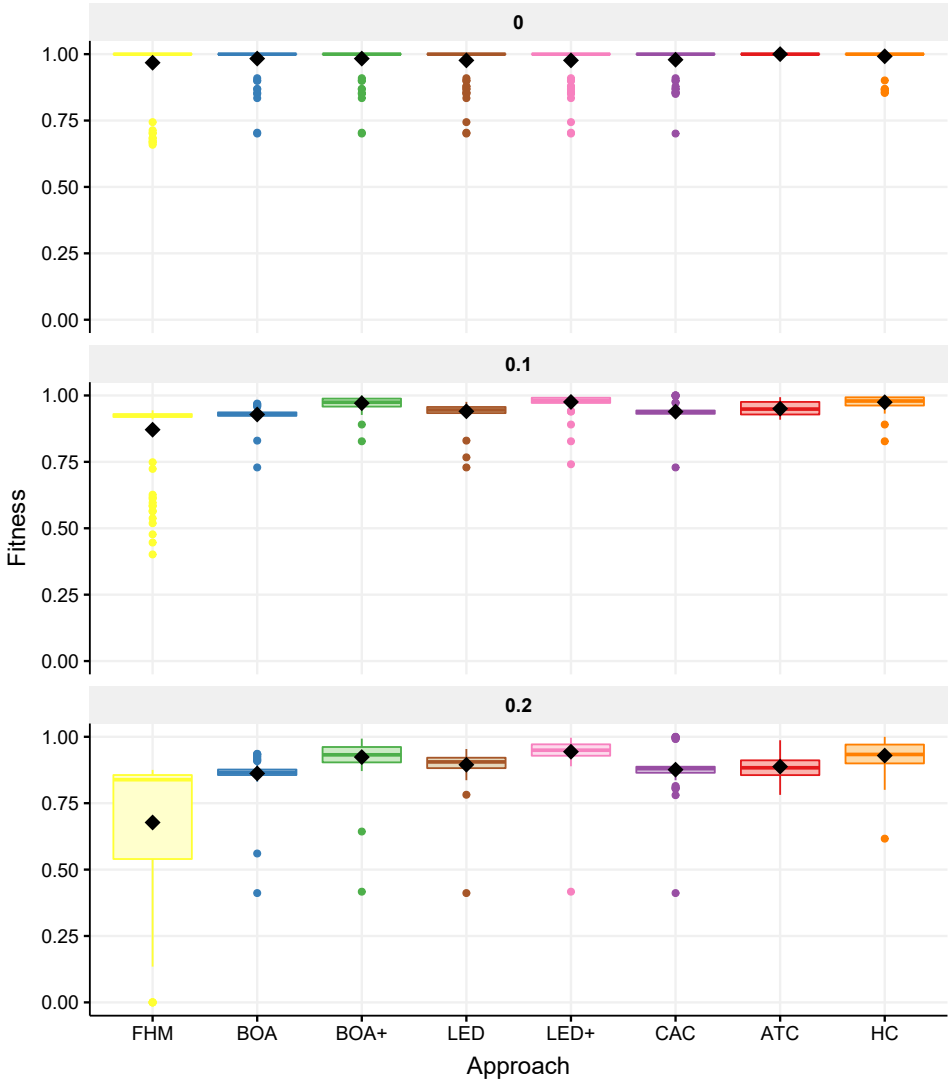


Figure C.1: Process model fitness after trace clustering aggregated over all synthetic event logs depending on the noise level.

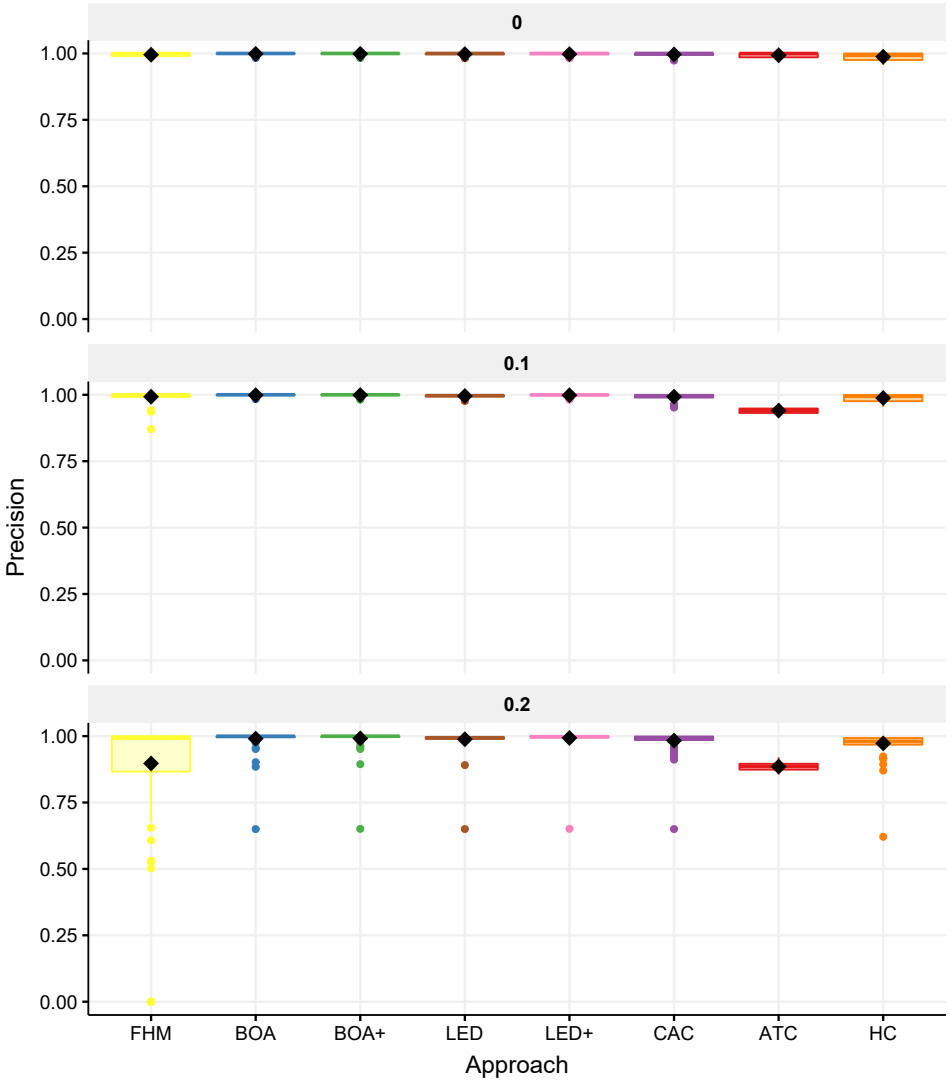


Figure C.2: Process model precision after trace clustering aggregated over all synthetic event logs depending on the noise level.

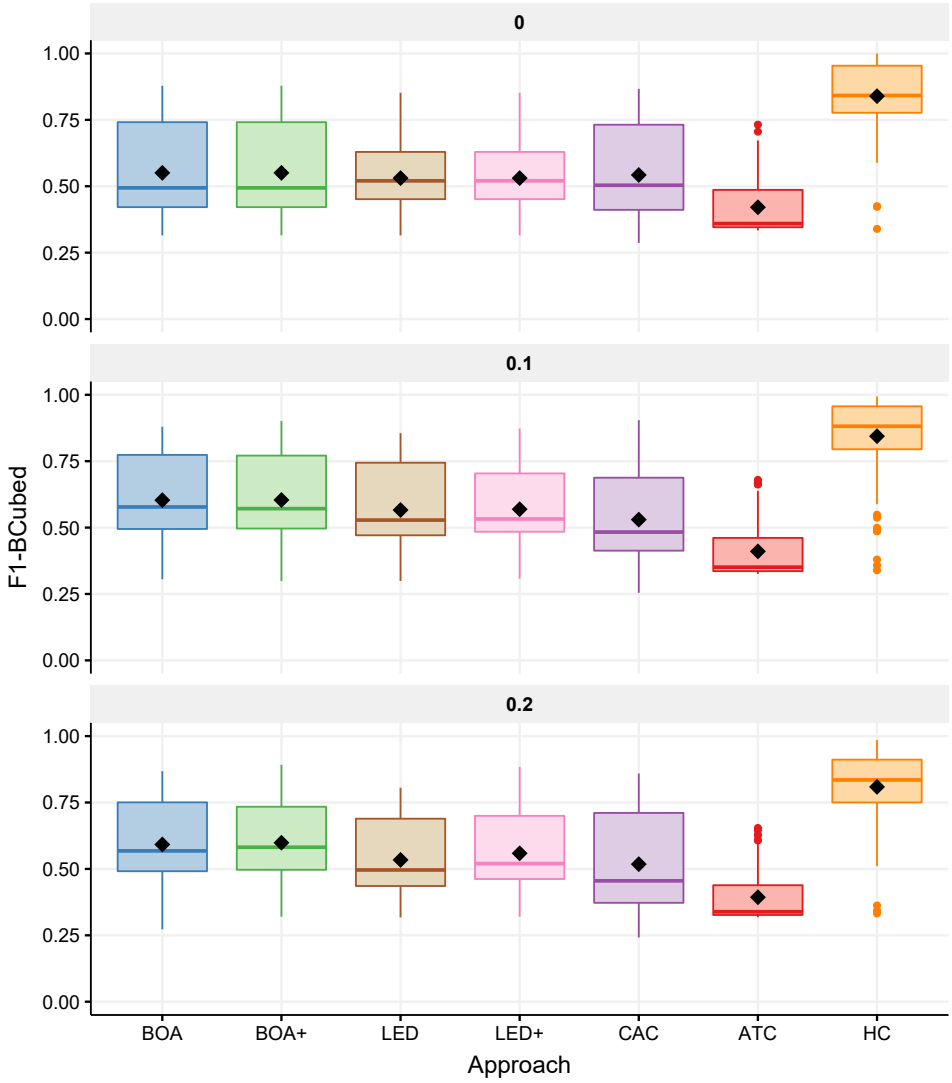


Figure C.3: F1-BCubed results after trace clustering aggregated over all synthetic event logs depending on the noise level.

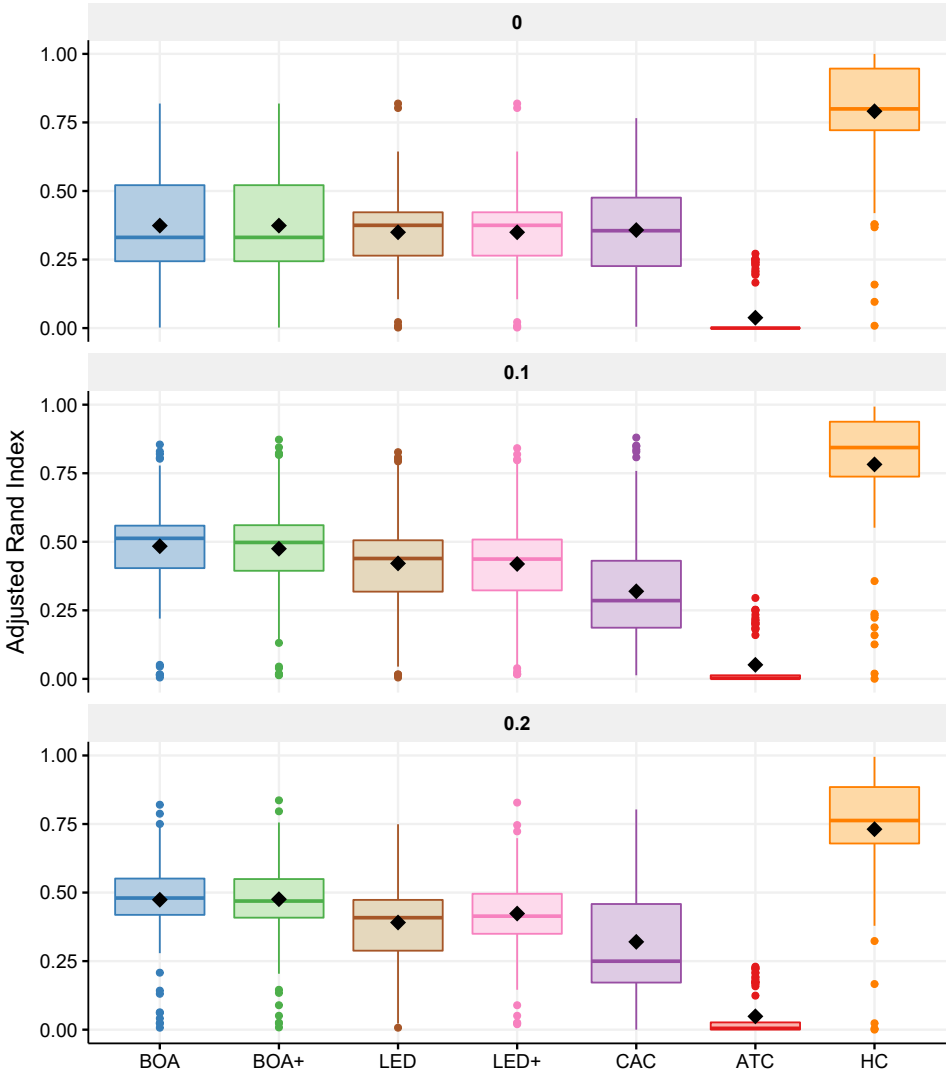


Figure C.4: Adjusted rand index results after trace clustering aggregated over all synthetic event logs depending on the noise level.

Bibliography

[Aal11] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-19345-3.

[Aal13] Wil M. P. van der Aalst. "Business Process Management: A Comprehensive Survey." In: *ISRN Software Engineering* 2013 (2013), pp. 1–37. DOI: 10.1155/2013/507984.

[Aal16] Wil M. P. van der Aalst. *Process Mining*. Springer Berlin Heidelberg, 2016. DOI: 10.1007/978-3-662-49851-4.

[AAD12] Wil M. P. van der Aalst, Arya Adriansyah, and B. F. van Dongen. "Replaying history on process models for conformance checking and performance analysis." In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.2 (2012), pp. 182–192. DOI: 10.1002/widm.1045.

[AAD11] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. "Causal Nets: A Modeling Language Tailored towards Process Discovery." In: *CONCUR 2011 – Concurrency Theory*. Ed. by Joost-Pieter Katoen and Barbara König. Springer Berlin Heidelberg, 2011, pp. 28–42. DOI: 10.1007/978-3-642-23217-6_3.

[ABD05] Wil M. P. van der Aalst, H. T. de Beer, and B. F. van Dongen. "Process Mining and Verification of Properties: An Approach Based on Temporal Logic." In: *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*. Springer Berlin Heidelberg, 2005, pp. 130–147. DOI: 10.1007/11575771_11.

[ABZ17] Wil M. P. van der Aalst, Alfredo Bolt, and Sebastiaan J. van Zelst. "RapidProM: Mine Your Processes and Not Just Your Data." In: *CoRR* abs/1703.03740 (2017). arXiv: 1703.03740.

[Aal+10] Wil M. P. van der Aalst, K. M. van Hee, J. M. van der Werf, and M. Verdonk. "Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor." In: *Computer* 43.3 (Mar. 2010), pp. 90–93. DOI: 10.1109/MC.2010.61.

[ALS16] Wil M. P. van der Aalst, Marcello La Rosa, and Flávia Maria Santoro. "Business Process Management: Don't Forget to Improve the Process!" In: *Business & Information Systems Engineering* 58.1 (Feb. 2016), pp. 1–6. DOI: 10.1007/s12599-015-0409-x.

[Aal+07] Wil M. P. van der Aalst, H. A. Reijers, A. J. M. M. Weijters, B. F. van Dongen, A. K. Alves de Medeiros, M. Song, and H. M. W. Verbeek. "Business process mining: An industrial application." In: *Information Systems* 32.5 (July 2007), pp. 713–732. DOI: 10.1016/j.is.2006.05.003.

[AWM04] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. "Workflow mining: Discovering process models from event logs." In: *IEEE Transactions on Knowledge and Data Engineering* 16.9 (2004), pp. 1128–1142. DOI: 10.1109/TKDE.2004.47.

[AS12] Rafael Accorsi and Thomas Stocker. "Discovering Workflow Changes with Time-Based Trace Clustering." In: *Data-Driven Process Discovery and Analysis*. Springer Berlin Heidelberg, 2012, pp. 154–168. DOI: 10.1007/978-3-642-34044-4_9.

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining Association Rules Between Sets of Items in Large Databases." In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. SIGMOD '93. New York, NY, USA: ACM Press, 1993, pp. 207–216. DOI: 10.1145/170035.170072.

[AM16] Annalisa Appice and Donato Malerba. "A Co-Training Strategy for Multiple View Clustering in Process Mining." In: *IEEE Transactions on Services Computing* 9.6 (2016), pp. 832–845. DOI: 10.1109/tsc.2015.2430327.

[Arz+13] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps." In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI '14*. PLDI '14. New York, NY, USA: ACM Press, 2013, pp. 259–269. DOI: 10.1145/2594291.2594299.

[Aug+17] Adriano Augusto, Raffaele Conforti, Marlon Dumas, and Marcello La Rosa. "Split Miner: Discovering Accurate and Simple Business Process Models from Event Logs." In: *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, Nov. 2017, pp. 1–10. DOI: 10.1109/ICDM.2017.9.

[ADW08] Ahmed Awad, Gero Decker, and Mathias Weske. "Efficient Compliance Checking Using BPMN-Q and Temporal Logic." In: *Business Process Management*. Cham: Springer Berlin Heidelberg, 2008, pp. 326–341. DOI: 10.1007/978-3-540-85758-7_24.

[AW09] Ahmed Awad and Mathias Weske. "Visualization of Compliance Violation in Business Process Models." In: *Business Process*

Management Workshops. Cham: Springer Berlin Heidelberg, 2009, pp. 182–193. DOI: 10.1007/978-3-642-12186-9_17.

[Bai+17] Thomas Baier, Claudio Di Ciccio, Jan Mendling, and Mathias Weske. "Matching events and activities by integrating behavioral aspects and label analysis." In: *Software & Systems Modeling* May (2017). DOI: 10.1007/s10270-017-0603-z.

[BMW14] Thomas Baier, Jan Mendling, and Mathias Weske. "Bridging abstraction layers in process mining." In: *Information Systems* 46 (Dec. 2014), pp. 123–139. DOI: 10.1016/j.is.2014.04.004.

[BSB17] Nithish Pai Ballambettu, Mahima Agumbe Suresh, and R. P. Jagadeesh Chandra Bose. "Analyzing Process Variants to Understand Differences in Key Performance Indices." In: *Advanced Information Systems Engineering*. Springer International Publishing, 2017, pp. 298–313. DOI: 10.1007/978-3-319-59536-8_19.

[Bec+14] Jörg Becker, Patrick Delfmann, Hanns Alexander Dietrich, Matthias Steinhorst, and Mathias Eggert. "Business process compliance checking - applying and evaluating a generic pattern matching approach for conceptual models in the financial sector." In: *Information Systems Frontiers* (2014), pp. 359–405. ISSN: 1387-3326. DOI: 10.1007/s10796-014-9529-y.

[Blu15] Fabian Rojas Blum. "Metrics in process discovery." In: *Technical Report TR/DCC-2015-6*. 2015.

[BLA18] Alfredo Bolt, Massimiliano de Leoni, and Wil M. P. van der Aalst. "Process variant comparison: Using event logs to detect differences in behavior and business rules." In: *Information Systems* 74 (May 2018), pp. 53–66. DOI: 10.1016/j.is.2017.12.006.

[BA09] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. "Context Aware Trace Clustering: Towards Improving Process Mining Results." In: *Proceedings of the 2009 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 2009, pp. 401–412. DOI: 10.1137/1.9781611972795.35.

[BA10] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. "Trace Clustering Based on Conserved Patterns: Towards Achieving Better Process Models." In: *Business Process Management Workshops*. Cham: Springer Berlin Heidelberg, 2010, pp. 170–181. DOI: 10.1007/978-3-642-12186-9_16.

[Bos+14] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indre Zliobaite, and Mykola Pechenizkiy. "Dealing With Concept Drifts in Process Mining." In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (Jan. 2014), pp. 154–171. DOI: 10.1109/tnnls.2013.2278313.

[Bos+11] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indrė Žliobaitė, and Mykola Pechenizkiy. "Handling Concept Drift in Process Mining." In: *Proceedings of the 23th International Conference on Advanced Information Systems Engineering*. Springer International Publishing, 2011, pp. 391–405. DOI: 10.1007/978-3-642-21640-4_30.

[BGW09] Melike Bozkaya, Joost Gabriels, and Jan Martijn van der Werf. "Process Diagnostics: A Method Based on Process Mining." In: *2009 International Conference on Information, Process, and Knowledge Management*. IEEE, Feb. 2009, pp. 22–27. DOI: 10.1109/eknow.2009.29.

[BK07] Daniel Bratton and James Kennedy. "Defining a Standard for Particle Swarm Optimization." In: *2007 IEEE Swarm Intelligence Symposium*. IEEE, Apr. 2007. DOI: 10.1109/sis.2007.368035.

[BDA14] J. C. A. M. Buijs, B. F. van Dongen, and Wil M. P. van der Aalst. "Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity." In: *International Journal of Cooperative Information Systems* 23.01 (Mar. 2014). DOI: 10.1142/s0218843014400012.

[Bur16] Andrea Burattin. "PLG2: Multiperspective process randomization with online and offline simulations." In: *CEUR Workshop Proceedings*. Vol. 1789. 2016, pp. 1–6.

[Bur+12] Andrea Burattin, Fabrizio M. Maggi, Wil M. P. van der Aalst, and Alessandro Sperduti. "Techniques for a Posteriori Analysis of Declarative Processes." In: *2012 IEEE 16th International Enterprise Distributed Object Computing Conference*. IEEE, Sept. 2012, pp. 41–50. DOI: 10.1109/edoc.2012.15.

[BS11] Andrea Burattin and Alessandro Sperduti. "PLG: A Framework for the Generation of Business Process Models and Their Execution Logs." In: *Business Process Management Workshops*. Cham: Springer Berlin Heidelberg, 2011, pp. 214–219. DOI: 10.1007/978-3-642-20511-8_20.

[Cal+17] Diego Calvanese, Tahir Emre Kalayci, Marco Montali, and Stefano Tinella. "Ontology-Based Data Access for Extracting Event Logs from Legacy Data: The onprom Tool and Methodology." In: *Business Information Systems*. Cham: Springer International Publishing, 2017, pp. 220–236. DOI: 10.1007/978-3-319-59336-4_16.

[CG12] Josep Carmona and Ricard Gavaldà. "Online Techniques for Dealing with Concept Drift in Process Mining." In: *Advances in Intelligent Data Analysis XI*. Springer Berlin Heidelberg, 2012, pp. 90–102. DOI: 10.1007/978-3-642-34156-4_10.

[CVB13] Filip Caron, Jan Vanthienen, and Bart Baesens. "Comprehensive rule-based compliance checking and risk management with process mining." In: *Decision Support Systems* 54.3 (Feb. 2013), pp. 1357–1369. DOI: 10.1016/j.dss.2012.12.012.

[Cas+04] Malu Castellanos, Fabio Casati, Umeshwar Dayal, and Ming-Chien Shan. "A Comprehensive and Automated Approach to Intelligent Business Processes Execution Analysis." In: *Distributed and Parallel Databases* 16.3 (Nov. 2004), pp. 239–273. DOI: 10.1023/b:dapd.0000031635.88567.65.

[CCD17] Thomas Chatain, Josep Carmona, and Boudewijn van Dongen. "Alignment-Based Trace Clustering." In: *Conceptual Modeling*. Springer International Publishing, 2017, pp. 295–308. DOI: 10.1007/978-3-319-69904-2_24.

[CYR09] Yang Chen, Jing Yang, and William Ribarsky. "Toward effective insight management in visual analytics systems." In: *2009 IEEE Pacific Visualization Symposium*. IEEE, Apr. 2009. DOI: 10.1109/pacificvis.2009.4906837.

[Che+02] Yixin Chen, Guozhu Dong, Jiawei Han, Benjamin W. Wah, and Jianyoung Wang. "Multi-Dimensional Regression Analysis of Time-Series Data Streams." In: *Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 323–334. DOI: 10.1016/b978-155860869-6/50036-6.

[CJ17] Tiffany Chiu and Mieke Julie Jans. "Process Mining of Event Logs: A Case Study Evaluating Internal Control Effectiveness." In: *SSRN Electronic Journal* (2017). DOI: 10.2139/ssrn.3136043.

[CK02] M. Clerc and J. Kennedy. "The particle swarm - explosion, stability, and convergence in a multidimensional complex space." In: *IEEE Transactions on Evolutionary Computation* 6.1 (2002), pp. 58–73. DOI: 10.1109/4235.985692.

[Coh88] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, 1988. ISBN: 978-0-8058-0283-2.

[Coh92] Jacob Cohen. "Statistical Power Analysis." In: *Current Directions in Psychological Science* 1.3 (June 1992), pp. 98–101. DOI: 10.1111/1467-8721.ep10768783.

[Con+16] Raffaele Conforti, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. "BPMN Miner: Automated discovery of BPMN process models with hierarchical structure." In: *Information Systems* 56 (Mar. 2016), pp. 284–303. DOI: 10.1016/j.is.2015.07.004.

[CW99] Jonathan E. Cook and Alexander L. Wolf. "Software process validation: quantitatively measuring the correspondence of a process to a model." In: *ACM Transactions on Software Engineering and Methodology* 8.2 (1999), pp. 147–176. DOI: 10.1145/304399.304401.

[CKO92] Bill Curtis, Marc I. Kellner, and Jim Over. "Process modeling." In: *Communications of the ACM* 35.9 (Sept. 1992), pp. 75–90. DOI: 10.1145/130994.130998.

[Dav85] Fred D. Davis. "A technology acceptance model for empirically testing new end-user information systems: Theory and results." PhD thesis. MIT, 1985.

[DBW18] Pieter De Koninck, Seppe vanden Broucke, and Jochen De Weerdt. "act2vec, trace2vec, log2vec, and model2vec: Representation Learning for Business Processes." In: *Business Process Management*. Cham: Springer International Publishing, 2018, pp. 305–321. DOI: 10.1007/978-3-319-98648-7_18.

[DD17] Pieter De Koninck and Jochen De Weerdt. "Similarity-Based Approaches for Determining the Number of Trace Clusters in Process Discovery." In: *Transactions on Petri Nets and Other Models of Concurrency XII*. Springer Berlin Heidelberg, 2017, pp. 19–42. DOI: 10.1007/978-3-662-55862-1_2.

[DDB16] Pieter De Koninck, Jochen De Weerdt, and Seppe K. L. M. vanden Broucke. "Explaining clusterings of process instances." In: *Data Mining and Knowledge Discovery* 31.3 (Dec. 2016), pp. 774–808. DOI: 10.1007/s10618-016-0488-4.

[De +17] Pieter De Koninck, Klaas Nelissen, Bart Baesens, Seppe vanden Broucke, Monique Snoeck, and Jochen De Weerdt. "An Approach for Incorporating Expert Knowledge in Trace Clustering." In: *Advanced Information Systems Engineering*. Springer International Publishing, 2017, pp. 561–576. DOI: 10.1007/978-3-319-59536-8_35.

[DMA12] Massimiliano De Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. "Aligning Event Logs and Declarative Process Models for Conformance Checking." In: *Business Process Management*. Cham: Springer Berlin Heidelberg, 2012, pp. 82–97. DOI: 10.1007/978-3-642-32885-5_6.

[Del+17] Pavlos Delias, Michael Doumpos, Evangelos Grigoroudis, and Nikolaos Matsatsinis. "A non-compensatory approach for trace clustering." In: *International Transactions in Operational Research* (2017). DOI: 10.1111/itor.12395.

[Dem+17] Çağatay Demiralp, Peter J. Haas, Srinivasan Parthasarathy, and Tejaswini Pedapati. "Foresight." In: *Proceedings of the VLDB Endowment*. Vol. 10. 12. VLDB Endowment, Aug. 2017, pp. 1937–1940. DOI: 10.14778/3137765.3137813.

[Dem06] Janez Demšar. "Statistical Comparisons of Classifiers over Multiple Data Sets." In: *Journal of Machine learning research* 7 (Dec. 2006), pp. 1–30. ISSN: 1532-4435.

[Di +15] Claudio Di Ciccio, Mitchel H. M. Schouten, Massimiliano de Leoni, and Jan Mendling. "Declarative process discovery with MINERful in ProM." In: *Proceedings of the Demo Session of the 13th International Conference on Business Process Management*. Vol. 1418. CEUR Workshop Proceedings. 2015, pp. 60–64.

[DW17] Remco Dijkman and Anna Wilbik. "Linguistic summarization of event logs – A practical approach." In: *Information Systems* 67 (July 2017), pp. 114–125. DOI: 10.1016/j.is.2017.03.009.

[Din+19] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. "QuickInsights." In: *Proceedings of the 2019 International Conference on Management of Data - SIGMOD '19*. ACM Press, 2019. DOI: 10.1145/3299869.3314037.

[Don17] B. F. van Dongen. "Dataset BPI Challenge 2017." In: *4TU.Centre for Research Data* (2017). DOI: 10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b.

[Don19] B. F. van Dongen. "Dataset BPI Challenge 2019." In: *4TU.Centre for Research Data* (2019). DOI: 10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1.

[Don+17] Boudewijn van Dongen, Josep Carmona, Thomas Chatain, and Farbod Taymouri. "Aligning Modeled and Observed Behavior: A Compromise Between Computation Complexity and Quality." In: *Advanced Information Systems Engineering*. Ed. by Eric Dubois and Klaus Pohl. Cham: Springer International Publishing, 2017, pp. 94–109. DOI: 10.1007/978-3-319-59536-8_7.

[Eck+15] Maikel L. van Eck, Xixi Lu, Sander J. J. Leemans, and Wil M. P. van der Aalst. "PM2: A Process Mining Project Methodology." In: *Advanced Information Systems Engineering*. Springer International Publishing, 2015, pp. 297–313. DOI: 10.1007/978-3-319-19069-3_19.

[Eka+13] Chathura C. Ekanayake, Marlon Dumas, Luciano Garcìa-Bañuelos, and Marcello La Rosa. "Slice, Mine and Dice: Complexity-Aware Automated Discovery of Business Process Models." In: *Business Process Management*. Springer Berlin Heidelberg, 2013, pp. 49–64. DOI: 10.1007/978-3-642-40176-3_6.

[ETF16] Joerg Evermann, Tom Thaler, and Peter Fettke. "Clustering Traces Using Sequence Alignment." In: *Business Process Management Workshops*. Cham: Springer International Publishing, 2016, pp. 179–190. DOI: 10.1007/978-3-319-42887-1_15.

[FZ14] Michael Fellmann and Andrea Zasada. "State-of-the-Art of Business Process Compliance Approaches : a Survey." In: *Proceedings of the 22th European Conference on Information Systems*. 2014, pp. 1–17.

[Fer09] Diogo R. Ferreira. "Applied Sequence Clustering Techniques for Process Mining." In: *Handbook of Research on Business Process Modeling* April (2009), pp. 481–502. DOI: 10.4018/978-1-60566-288-6.ch022.

[Fri37] Milton Friedman. "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance." In: *Journal of the American Statistical Association* 32.200 (1937), pp. 675–701. DOI: 10.1080/01621459.1937.10503522.

[GH06] Liqiang Geng and Howard J. Hamilton. "Interestingness measures for data mining." In: *ACM Computing Surveys* 38.3 (Sept. 2006). DOI: 10.1145/1132960.1132963.

[GCC09] Kerstin Gerke, Jorge Cardoso, and Alexander Claus. "Measuring the compliance of processes with reference models." In: *On the Move to Meaningful Internet Systems: OTM 2009*. Springer Berlin Heidelberg, 2009, pp. 76–93. DOI: 10.1007/978-3-642-05148-7_8.

[GT09] Kerstin Gerke and Gerrit Tamm. "Continuous Quality Improvement of IT Processes based on Reference Models and Process Mining." In: *Americas Conference on Information Systems*. 2009.

[GA19] Mahdi Ghasemi and Daniel Amyot. "From event logs to goals: a systematic literature review of goal-oriented process mining." In: *Requirements Engineering* 25.1 (Jan. 2019), pp. 67–93. DOI: 10.1007/s00766-018-00308-3.

[GHV08] Stijn Goedertier, Raf Haesen, and Jan Vanthienen. "Rule-based business process modelling and enactment." In: *International Journal of Business Process Integration and Management* 3.3 (2008), p. 194. DOI: 10.1504/ijbpim.2008.023219.

[Goe+11] Stijn Goedertier, Jochen De Weerdt, David Martens, Jan Vanthienen, and Bart Baesens. "Process discovery in event logs: An application in the telecom industry." In: *Applied Soft Computing* 11.2 (Mar. 2011), pp. 1697–1710. DOI: 10.1016/j.asoc.2010.04.025.

[GR08] Guido Governatori and Antonino Rotolo. "An algorithm for business process compliance." In: *Frontiers in Artificial Intelligence and Applications* 189.1 (2008), pp. 186–191. DOI: 10.3233/978-1-58603-952-3-186.

[GZ05] Gösta Grahne and Jianfei Zhu. "Fast Algorithms for Frequent Itemset Mining using FP-trees." In: *IEEE Transactions on Knowledge and Data Engineering* 17.10 (Oct. 2005), pp. 1347–1362. DOI: 10.1109/tkde.2005.166.

[Gre+06] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. "Discovering Expressive Process Models by Clustering Log Traces." In: *IEEE Transactions on Knowledge and Data Engineering* 18.8 (Aug. 2006), pp. 1010–1027. DOI: 10.1109/tkde.2006.123.

[Gua+11] Salvatore Guarnieri, Marco Pistoia, Omer Tripp, Julian Dolby, Stephen Teilhet, and Ryan Berg. "Saving the World Wide Web from Vulnerable JavaScript." In: *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. ISSTA '11. New York, NY, USA: ACM Press, 2011, pp. 177–187. DOI: 10.1145/2001420.2001442.

[GK02] A. Gunasekaran and B. Kobu. "Modelling and analysis of business process reengineering." In: *International Journal of Production Research* 40.11 (Jan. 2002), pp. 2521–2546. DOI: 10.1080/00207540210132733.

[Gün+06] Christian W. Günther, Stefanie Rinderle, Manfred Reichert, and Wil M. P. van der Aalst. "Change Mining in Adaptive Process Management Systems." In: *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*. Springer Berlin Heidelberg, 2006, pp. 309–326. DOI: 10.1007/11914853_19.

[Gün+08] Christian W. Günther, Stefanie Rinderle-Ma, Manfred Reichert, Wil M. P. van der Aalst, and Jan C. Recker. "Using process mining to learn from process changes in evolutionary systems." In: *International Journal of Business Process Integration and Management (IJBPIM)* 3.1 (2008), pp. 61–79. DOI: 10.1504/IJBPIM.2008.019348.

[HKP11] Jiawei Han, Micheline Kamber, and Jain Pei. *Data Mining – Concepts & Techniques*. 3rd ed. Waltham, USA: Morgan Kaufmann Publishers, 2011, pp. 1–744. ISBN: 9780123814791. DOI: 10.1016/B978-0-12-381479-1.00001-0.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data - SIGMOD '00*. New York, NY, USA: ACM Press, 2000. DOI: 10.1145/342009.335372.

[Has+18] Mustafa Hashmi, Guido Governatori, Ho-Pun Lam, and Moe Thandar Wynn. "Are we done with business process compliance: state of the art and challenges ahead." In: *Knowledge and Information Systems* 57.1 (Jan. 2018), pp. 79–133. DOI: 10.1007/s10115-017-1142-1.

[Ho05] Shen-Shyang Ho. "A Martingale Framework for Concept Change Detection in Time-Varying Data Streams." In: *Proceedings of the 22nd international conference on Machine learning - ICML '05*. 2004. ACM Press, 2005, pp. 321–327. DOI: 10.1145/1102351.1102392.

[HB15] B. F. A. Hompes and J. Buijs. "Discovering Deviating Cases and Process Variants Using Trace Clustering." In: *Proceedings of the 27th Benelux Conference on Artificial Intelligence (BNAIC)*. 2015.

[HBA15] Bart F. A. Hompes, Joos C. A. M. Buijs, and Wil M. P. van der Aalst. "Detecting Change in Processes Using Comparative Trace Clustering." In: *Proceedings of the 5th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2015)*. 2015, pp. 95–108.

[Hop19] Brian Hopkins. "The Forrester Wave™: Enterprise Insight Platforms, Q1 2019." In: *Forrester* (2019).

[HA85] Lawrence Hubert and Phipps Arabie. "Comparing partitions." In: *Journal of Classification* 2.1 (Dec. 1985), pp. 193–218. DOI: 10.1007/BF01908075.

[IEE11] IEEE Task Force on Process Mining. "Process Mining Manifesto." In: *Business Process Management Workshops*. Cham, 2011, pp. 169–194. DOI: 10.1007/978-3-642-28108-2_19.

[JB96] Stefan Jablonski and Christoph Bussler. *Workflow management: modeling concepts, architecture and implementation*. Vol. 392. International Thomson Computer Press London, 1996.

[Jab+19] Stefan Jablonski, Maximilian Rögliner, Stefan Schönig, and Katrin Maria Wyrtki. "Multi-Perspective Clustering of Process Execution Traces." In: *Enterprise Modelling and Information Systems Architectures (EMISAJ)* (2019). DOI: 10.18417/emisa.14.2.

[JA12] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. "Process diagnostics using trace alignment: Opportunities, issues, and challenges." In: *Information Systems* 37.2 (Apr. 2012), pp. 117–141. ISSN: 0306-4379. DOI: 10.1016/j.is.2011.08.003.

[JGP19] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. "Interactive Data Exploration with Smart Drill-Down." In: *IEEE Transactions on Knowledge and Data Engineering* 31.1 (Jan. 2019), pp. 46–60. DOI: 10.1109/tkde.2017.2685998.

[KE95] J. Kennedy and R. Eberhart. "Particle swarm optimization." In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. IEEE, 1995. DOI: 10.1109/icnn.1995.488968.

[Key+12] Alicia Key, Bill Howe, Daniel Perry, and Cecilia Aragon. "Vizdeck: Self-organizing dashboards for visual analytics." In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (demo)*. ACM. 2012, pp. 681–684.

[KC11] Ralph Kimball and Joe Caserta. *The Data WarehouseETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Wiley Publishing, Inc., 2011.

[KMW19] Christopher Klinkmüller, Richard Müller, and Ingo Weber. "Mining Process Mining Practices: An Exploratory Characterization of Information Needs in Process Analytics." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 322–337. DOI: 10.1007/978-3-030-26619-6_21.

[Knu+10] David Knuplesch, Linh Thao Ly, Stefanie Rinderle-Ma, Holger Pfeifer, and Peter Dadam. "On Enabling Data-Aware Compliance Checking of Business Process Models." In: *Conceptual Modeling – ER 2010*. Springer Berlin Heidelberg, 2010, pp. 332–346. DOI: 10.1007/978-3-642-16373-9_24.

[KR13] Agnes Koschmider and Hajo A. Reijers. "Improving the process of process modelling by the use of domain process patterns." In: *Enterprise Information Systems* 9.1 (Dec. 2013), pp. 29–57. DOI: 10.1080/17517575.2013.857792.

[Kru+11] L. Krumov, C. Fretter, M. Müller-Hannemann, K. Weihe, and M. T. Hütt. "Motifs in co-authorship networks and their relation to the impact of scientific publications." In: *European Physical Journal B* 84.4 (2011), pp. 535–540. ISSN: 1434-6028. DOI: 10.1140/epjb/e2011-10746-5.

[Kru+10] Lachezar Krumov, Immanuel Schweizer, Dirk Bradler, and Thorsten Strufe. "Leveraging network motifs for the adaptation of structured peer-to-peer-networks." In: *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*. IEEE, Dec. 2010, pp. 1–5. DOI: 10.1109/GLOCOM.2010.5683139.

[LKD11] Geetika T. Lakshmanan, Paul T. Keyser, and Songyun Duan. "Detecting Changes in a Semi-Structured Business Process through Spectral Graph Analysis." In: *2011 IEEE 27th International Conference on Data Engineering Workshops*. IEEE, Apr. 2011, pp. 255–260. DOI: 10.1109/ICDEW.2011.5767640.

[LHS08] Bettina Laugwitz, Theo Held, and Martin Schrepp. "Construction and Evaluation of a User Experience Questionnaire." In: *HCI and Usability for Education and Work*. Springer Berlin Heidelberg, 2008, pp. 63–76. DOI: 10.1007/978-3-540-89350-9_6.

[LFA14a] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. "Process and Deviation Exploration with Inductive visual Miner." In: vol. 1295. 2014, pp. 46–50.

[LFA14b] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. "Process and Deviation Exploration with Inductive visual Miner." In: *CEUR Workshop Proceedings* 1295 (2014), pp. 46–50. ISSN: 1613-0073.

[LPW19] Sander J. J. Leemans, Erik Poppe, and Moe T. Wynn. "Directly Follows-Based Process Mining: a Tool." In: *Proceedings of the ICPM Demo Track 2019* 2374 (June 2019).

[Lev66] Vladimir I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals." In: *Soviet Physics Doklady* 10.8 (Feb. 1966), pp. 707–710.

[LL14] Innar Liiv and Ott Lepik. "Business Process Mining in Warehouses: a Case Study." In: *Proceedings of the 11th International Baltic Conference*. 2014, pp. 387–394.

[Lij+16] Jefrey Lijffijt, Bo Kang, Kai Puolamäki, and Tijl De Bie. "SIDE: A Web App for Interactive Visual Data Exploration with Subjective Feedback." In: *KDD IDEA Workshop*. 2016, pp. 86–95.

[Lin91] J. Lin. "Divergence measures based on the Shannon entropy." In: *IEEE Transactions on Information Theory* 37.1 (1991), pp. 145–151. DOI: 10.1109/18.61115.

[LF19] Iezalde F. Lopes and Diogo R. Ferreira. "A Survey of Process Mining Competitions: the BPI Challenges 2011-2018." In: *Business Process Management Workshops*. 2019.

[LS12] Daniela Luengo and Marcos Sepúlveda. "Applying Clustering in Process Mining to Find Different Versions of a Business Process That Changes over Time." In: *Business Process Management Workshops*. Cham: Springer Berlin Heidelberg, 2012, pp. 153–158. DOI: 10.1007/978-3-642-28108-2_15.

[Luo+18] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. "DeepEye: Towards Automatic Data Visualization." In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, Apr. 2018. DOI: 10.1109/icde.2018.00019.

[LRD10] Linh Thao Ly, Stefanie Rinderle-Ma, and Peter Dadam. "Design and Verification of Instantiable Compliance Rule Graphs in Process-Aware Information Systems." In: *Advanced Information Systems Engineering*. Springer International Publishing, 2010, pp. 9–23. DOI: 10.1007/978-3-642-13094-6_3.

[Maa+15] Abderrahmane Maaradji, Marlon Dumas, Marcello La Rosa, and Alireza Ostovar. "Fast and Accurate Business Process Drift Detection." In: *Business Process Management*. Cham: Springer International Publishing, 2015, pp. 406–422. DOI: 10.1007/978-3-319-23063-4_27.

[MBA13] Fabrizio M. Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. "A Knowledge-Based Integrated Approach for Discovering and Repairing Declare Maps." In: *Advanced Information Systems Engineering*. Springer Berlin Heidelberg, 2013, pp. 433–448. DOI: 10.1007/978-3-642-38709-8_28.

[Mag+11] Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil M. P. van der Aalst. "Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata." In: *Business Process Management*. Cham: Springer Berlin Heidelberg, 2011, pp. 132–147. DOI: 10.1007/978-3-642-23059-2_13.

[Man16] Felix Mannhardt. "XESlite - Managing Large XES Event Logs in ProM." In: *BPM Center Report BPM-16-04* (2016). URL: http://bpmcenter.org/wp-content/uploads/reports/2016/BPM-16-04.pdf.

[Man18] Felix Mannhardt. "Multi-perspective Process Mining." PhD thesis. Technische Universiteit Eindhoven, 2018, p. 425. ISBN: 978-90-386-4438-7.

[MDR17] Felix Mannhardt, Massimiliano De Leoni, and Hajo A. Reijers. "Heuristic Mining Revamped: An Interactive, Data-aware, and Conformance-aware Miner." In: *CEUR Workshop Proceedings*. Vol. 1920. 2017.

[Man+15] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M. P. van der Aalst. "Balanced multi-perspective checking of process conformance." In: *Computing* 98.4 (Feb. 2015), pp. 407–437. DOI: 10.1007/s00607-015-0441-1.

[Man+16] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, Wil M. P. van der Aalst, and Pieter J. Toussaint. "From Low-Level Events to Activities - A Pattern-Based Approach." In: *International Conference on Business Process Management*. Vol. 2. 2016, pp. 125–141. DOI: 10.1007/978-3-319-45348-4_8.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Vol. 39. Cambridge University Press, Oct. 8, 2008. 496 pp. ISBN: 0521865719.

[MTA15] M. V. Manoj Kumar, Likewin Thomas, and B. Annappa. "Capturing the Sudden Concept Drift in Process Mining." In: *CEUR Workshop Proceedings*. Vol. 1371. 2015, pp. 132–143.

[Man+08] R. S. Mans, M. H. Schonenberg, M. Song, and Wil M. P. van der Aalst. "PROCESS MINING IN HEALTHCARE - A Case Study." In: *Proceedings of the First International Conference on Health Informatics*. SciTePress - Science, 2008. DOI: 10.5220/0001039201180125.

[MRO09] Selma Limam Mansar, Hajo A. Reijers, and Fouzia Ounnar. "Development of a decision-making strategy to improve the efficiency of BPR." In: *Expert Systems with Applications* 36.2 (Mar. 2009), pp. 3248–3262. DOI: 10.1016/j.eswa.2008.01.008.

[MBA15] J. Martjushev, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. "Change Point Detection and Dealing with Gradual and Multi-order Dynamics in Process Mining." In: *Perspectives in Business Informatics Research*. Springer International Publishing, 2015, pp. 161–178. DOI: 10.1007/978-3-319-21915-8_11.

[Mas51] Frank J. Massey. "The Kolmogorov-Smirnov Test for Goodness of Fit." In: *Journal of the American Statistical Association* 46.253 (Mar. 1951), pp. 68–78. DOI: 10.1080/01621459.1951.10500769.

[May00] Philipp Mayring. "Qualitative Content Analysis." In: *Forum: Qualitative Social Research* 1.2 (June 2000).

[McD09] John H. McDonald. "Handbook of Biological Statistics." In: *Sparky House Publishing* (2009), p. 291. DOI: 10.1017/CBO9781107415324.004.

[MRC07] Jan Mendling, Hajo A. Reijers, and Jorge Cardoso. "What Makes Process Models Understandable?" In: *Business Process Management*. Cham: Springer Berlin Heidelberg, 2007, pp. 48–63. DOI: 10.1007/978-3-540-75183-0_4.

[Mil+02] R. Milo, S. Shen-Orr, S. Itzkovitz, and N. Kashtan. "Network Motif: Simple Building Blocks of Complex Networks." In: *Science* 298.5594 (2002), pp. 824–827. DOI: 10.1126/science.298.5594.824.

[MS18] Tova Milo and Amit Somech. "Next-Step Suggestions for Modern Interactive Data Analysis Platforms." In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '18*. ACM Press, 2018. DOI: 10.1145/3219819.3219848.

[MC10] Jorge Muñoz-Gama and Josep Carmona. "A Fresh Look at Precision in Process Conformance." In: *Business Process Management*. Cham: Springer Berlin Heidelberg, 2010, pp. 211–226. DOI: 10.1007/978-3-642-15618-2_16.

[MC12] Jorge Muñoz-Gama and Josep Carmona. "A general framework for precision checking." In: *International Journal of Innovative Computing, Information and Control* 8.7 B (2012), pp. 5317–5339. ISSN: 1349-4198.

[MCA13] Jorge Muñoz-Gama, Josep Carmona, and Wil M. P. van der Aalst. "Conformance Checking in the Large: Partitioning and Topology." In: *Business Process Management*. Springer Berlin Heidelberg, 2013, pp. 130–145. DOI: 10.1007/978-3-642-40176-3_11.

[MRA18] Eduardo González López de Murillas, Hajo A. Reijers, and Wil M. P. van der Aalst. "Connecting databases with process mining: a meta model and toolset." In: *Software & Systems Modeling* 18.2 (Feb. 2018), pp. 1209–1247. DOI: 10.1007/s10270-018-0664-7.

[MVT16] Belgin Mutlu, Eduardo Veas, and Christoph Trattner. "VizRec." In: *ACM Transactions on Interactive Intelligent Systems* 6.4 (Nov. 2016), pp. 1–39. DOI: 10.1145/2983923.

[Ngu+14] Tien T. Nguyen, Pik-Mai Hui, F. Maxwell Harper, Loren Terveen, and Joseph A. Konstan. "Exploring the filter bubble." In: *Proceedings of the 23rd international conference on World wide web - WWW '14*. New York, NY, USA: ACM Press, 2014. DOI: 10.1145/2566486.2568012.

[Nie15] Florian Niedermann. "Deep Business Optimization." PhD thesis. 2015.

[NRM10a] Florian Niedermann, Sylvia Radeschutz, and Bernhard Mitschang. "Design-Time Process Optimization through Optimization Patterns and Process Model Matching." In: *2010 IEEE 12th Conference on Commerce and Enterprise Computing*. IEEE, Nov. 2010. DOI: 10.1109/cec.2010.9.

[NRM10b] Florian Niedermann, Sylvia Radeschütz, and Bernhard Mitschang. "Deep Business Optimization: A Platform for Automated Process Optimization." In: *ISSS/BPSC*. 2010.

[NSM16] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. "Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders." In: *Discovery Science*. Springer International Publishing, 2016, pp. 442–456. DOI: 10.1007/978-3-319-46307-0_28.

[NDF13] Erik H. J. Nooijen, Boudewijn F. van Dongen, and Dirk Fahland. "Automatic Discovery of Data-Centric and Artifact-Centric Processes." In: *Business Process Management Workshops*. Cham: Springer Berlin Heidelberg, 2013, pp. 316–327. DOI: 10.1007/978-3-642-36285-9_36.

[Ost+16] Alireza Ostovar, Abderrahmane Maaradji, Marcello La Rosa, Arthur H. M. ter Hofstede, and Boudewijn F. van Dongen. "Detecting Drift from Event Streams of Unpredictable Business Processes." In: *Conceptual Modeling*. Springer International Publishing, 2016, pp. 330–346. DOI: 10.1007/978-3-319-46397-1_26.

[Pap+16] Dimitra Papadimitriou, Georgia Koutrika, John Mylopoulos, and Yannis Velegrakis. "The Goal Behind the Action." In: *ACM Transactions on Database Systems* 41.4 (Dec. 2016), pp. 1–43. DOI: 10.1145/2934666.

[Ped+08] Carlos Pedrinaci, Dave Lambert, Branimir Wetzstein, Tammo van Lessen, Luchesar Cekov, and Marin Dimitrov. "SENTINEL: : a semantic business process monitoring tool." In: *Proceedings of the first international workshop on Ontology-supported business intelligence - OBI '08*. ACM Press, 2008. DOI: 10.1145/1452567.1452568.

[PSA07] Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. "DECLARE: Full Support for Loosely-Structured Processes." In: *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*. IEEE, Oct. 2007. DOI: 10.1109/edoc.2007.4384001.

[QYC12] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. "Diversifying top-k results." In: *Proceedings of the VLDB Endowment*. Vol. 5. 11. VLDB Endowment, July 2012, pp. 1124–1135. DOI: 10.14778/2350229.2350233.

[RFA12] Elham Ramezani, Dirk Fahland, and Wil M. P. van der Aalst. "Where Did I Misbehave? Diagnostic Information in Compliance Checking." In: *Business Process Management*. Cham: Springer Berlin Heidelberg, 2012, pp. 262–278. DOI: 10.1007/978-3-642-32885-5_21.

[Ran71] William M. Rand. "Objective Criteria for the Evaluation of Clustering Methods." In: *Journal of the American Statistical Association* 66.336 (1971), pp. 846–850.

[RW12] Manfred Reichert and Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems*. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-30409-5.

[RL05] H. Reijters and S. Liman Mansar. "Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics." In: *Omega* 33.4 (Aug. 2005), pp. 283–306. DOI: 10.1016/j.omega.2004.04.012.

[RHS95] Thomas Reps, Susan Horwitz, and Mooly Sagiv. "Precise interprocedural dataflow analysis via graph reachability." In: *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '95*. New York, New York, USA: ACM Press, 1995, pp. 49–61. DOI: 10.1145/199448.199462.

[RS17] Florian Richter and Thomas Seidl. "TESSERACT: Time-Drifts in Event Streams Using Series of Evolving Rolling Averages of Completion Times." In: *Business Process Management*. Cham: Springer International Publishing, 2017, pp. 289–305. DOI: 10.1007/978-3-319-65000-5_17.

[RRD04] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. "Correctness criteria for dynamic changes in workflow systems—a survey." In: *Data & Knowledge Engineering* 50.1 (July 2004), pp. 9–34. DOI: 10.1016/j.datak.2004.01.002.

[Río+13] Adela del Río-Ortega, Manuel Resinas, Cristina Cabanillas, and Antonio Ruiz-Cortés. "On the definition and design-time analysis of process performance indicators." In: *Information Systems* 38.4 (June 2013), pp. 470–490. DOI: 10.1016/j.is.2012.11.004.

[Ros+11] Marcello La Rosa, Hajo A. Reijers, Wil M. P. van der Aalst, Remco M. Dijkman, Jan Mendling, Marlon Dumas, and Luciano García-Bañuelos. "APROMORE: An advanced process model repository." In: *Expert Systems with Applications* 38.6 (June 2011), pp. 7029–7040. DOI: 10.1016/j.eswa.2010.12.012.

[Rou87] Peter J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis." In: *Journal of Computational and Applied Mathematics* 20.4 (Nov. 1987), pp. 53–65. ISSN: 0377-0427. DOI: 10.1016/0377-0427(87)90125-7.

[San+96] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. "Role-based access control models." In: *Computer* 29.2 (1996), pp. 38–47. DOI: 10.1109/2.485845.

[San+17] Mohammadreza Fani Sani, Wil van der Aalst, Alfredo Bolt, and Javier García-Algarra. "Subgroup Discovery in Process Mining." In: *Business Information Systems*. Cham: Springer International Publishing, 2017, pp. 237–252. DOI: 10.1007/978-3-319-59336-4_17.

[SAM98] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. "Discovery-driven exploration of OLAP data cubes." In: *Proceedings of the International Conference on Extending Database Technology*. Springer Berlin Heidelberg, 1998, pp. 168–182. DOI: 10.1007/bfb0100984.

[SO02] Paul Sarbanes and Michael Oxley. "Sarbanes-Oxley Act." In: *Washington DC* (2002).

[SLB03] Josef Schiefer, Beate List, and Robert M. Bruckner. "Process Data Store: A Real-Time Data Store for Monitoring Business Processes." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 760–770. DOI: 10.1007/978-3-540-45227-0_74.

[Sch15] Stefan Schönig. "SQL Queries for Declarative Process Mining on Event Logs of Relational Databases." In: 612052 (2015).

[SS10] Falk Schreiber and H. Schwöbbermeyer. "Motifs in biological networks." In: *Statistical and Evolutionary Analysis of Biological Networks* (2010), pp. 45–64.

[See+19a] Alexander Seeliger, Alejandro Sánchez Guinea, Timo Nolle, and Max Mühlhäuser. "ProcessExplorer: Intelligent Process Mining Guidance." In: *Business Process Management*. Cham: Springer International Publishing, 2019.

[SNM17] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. "Detecting Concept Drift in Processes using Graph Metrics on Process Graphs." In: *Proceedings of the 9th Conference on Subject-oriented Business Process Management - S-BPM ONE '17*. ACM Press, 2017. DOI: 10.1145/3040565.3040566.

[SNM18a] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. "Finding Structure in the Unstructured: Hybrid Feature Set Clustering for Process Discovery." In: *Business Process Management*. Cham: Springer International Publishing, 2018, pp. 288–304. DOI: 10.1007/978-3-319-98648-7_17.

[SNM18b] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. "Process Explorer: An Interactive Visual Recommendation System for Process Mining." In: *KDD Workshop on Interactive Data Exploration and Analytics* (2018).

[See+16a] Alexander Seeliger, Timo Nolle, Benedikt Schmidt, and Max Mühlhäuser. "Process Compliance Checking using Taint Flow Analysis." In: *Proceedings of the 37th International Conference on Information Systems - ICIS '16*. 2016.

[See+19b] Alexander Seeliger, Maximilian Ratzke, Timo Nolle, and Max Mühlhäuser. "ProcessExplorer: Interactive Visual Exploration of Event Logs with Analysis Guidance." In: *Proceedings of the 1st International Conference on Process Mining - ICPM'19 - Demo*. 2019.

[See+16b] Alexander Seeliger, Benedikt Schmidt, Immanuel Schweizer, and Max Mühlhäuser. "What Belongs Together Comes Together. Activity-centric Document Clustering for Information Work." In: *Proceedings of the 21st International Conference on Intelligent User Interfaces - IUI '16*. New York, NY, USA: ACM Press, 2016, pp. 60–70. DOI: 10.1145/2856767.2856777.

[SSM18] Alexander Seeliger, Michael Stein, and Max Mühlhäuser. "Can We Find Better Process Models? Process Model Improvement Using Motif-Based Graph Adaptation." In: *Business Process Management Workshops*. Cham: Springer International Publishing, 2018, pp. 230–242. DOI: 10.1007/978-3-319-74030-0_17.

[ST96] A. Silberschatz and A. Tuzhilin. "What makes patterns interesting in knowledge discovery systems." In: *IEEE Transactions on Knowledge and Data Engineering* 8.6 (1996), pp. 970–974. DOI: 10.1109/69.553165.

[SCJ16] Manish Singh, Michael J. Cafarella, and H. V. Jagadish. "DBExplorer: Exploratory Search in Databases." In: *EDBT*. 2016, pp. 89–100. ISBN: 9783893180707.

[Son+13] M. Song, H. Yang, S. H. Siadat, and M. Pechenizkiy. "A comparative study of dimensionality reduction techniques to enhance trace clustering performances." In: *Expert Systems with Applications* 40.9 (2013), pp. 3722–3737. DOI: 10.1016/j.eswa.2012.12.078.

[SGA09] Minseok Song, Christian W. Günther, and Wil M. P. van der Aalst. "Trace Clustering in Process Mining." In: *Business Process Management Workshops*. Cham: Springer, 2009, pp. 109–120. DOI: 10.1007/978-3-642-00328-8_11.

[Ste+17] Michael Stein, Karsten Weihe, Augustin Wilberg, Roland Kluge, Julian M. Klomp, Mathias Schnee, Lin Wang, and Max Mühlhäuser. "Distributed Graph-based Topology Adaptation using Motif Signatures." In: *ACM-SIAM Meeting on Algorithm Engineering & Experiments* (2017).

[SF12] Gail M. Sullivan and Richard Feinn. "Using Effect Size - or Why the P Value Is Not Enough." In: *Journal of Graduate Medical Education* 4.3 (Sept. 2012), pp. 279–282. DOI: 10.4300/jgme-d-12-00156.1.

[SB15] Yaguang Sun and Bernhard Bauer. "A Novel Top-Down Approach for Clustering Traces." In: *Advanced Information Systems Engineering*. Springer International Publishing, 2015, pp. 331–345. DOI: 10.1007/978-3-319-19069-3_21.

[SB16] Yaguang Sun and Bernhard Bauer. "A Novel Heuristic Method for Improving the Fitness of Mined Business Process Models." In: *Service-Oriented Computing*. Springer International Publishing, 2016, pp. 537–546. DOI: 10.1007/978-3-319-46295-0_33.

[SBW17] Yaguang Sun, Bernhard Bauer, and Matthias Weidlich. "Compound Trace Clustering to Generate Accurate and Simple Sub-Process Models." In: *Service-Oriented Computing*. Springer International Publishing, 2017, pp. 175–190. DOI: 10.1007/978-3-319-69035-3_12.

[Tan16] Colin Tankard. "What the GDPR means for businesses." In: *Network Security* 2016.6 (June 2016), pp. 5–8. DOI: 10.1016/s1353-4858(16)30056-3.

[Tha+15] Tom Thaler, Simon Ternis, Peter Fettke, and Peter Loos. "A Comparative Analysis of Process Instance Cluster Techniques." In: *Proceedings der 12. Internationalen Tagung Wirtschaftsinformatik (WI 2015)* August (2015), pp. 423–437.

[Van+15] Rob J. B. Vanwersch, Khurram Shahzad, Irene Vanderfeesten, Kris Vanhaecht, Paul Grefen, Liliane Pintelon, Jan Mendling, Godefridus G. van Merode, and Hajo A. Reijers. "A Critical Evaluation and Framework of Business Process Improvement Methods." In: *Business & Information Systems Engineering* 58.1 (Nov. 2015), pp. 43–53. DOI: 10.1007/s12599-015-0417-x.

[Var+15] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. "SeeDB." In: *Proceedings of the VLDB Endowment*. Vol. 8. 13. VLDB Endowment, 2015, pp. 2182–2193.

[Váz+16] Borja Vázquez-Barreiros, David Chapela, Manuel Mucientes, Manuel Lama, and Diego Berea. "Process mining in IT service management: A case study." In: *CEUR Workshop Proceedings*. Vol. 1592. 2016, pp. 16–30.

[VF10] Gabriel M. Veiga and Diogo R. Ferreira. "Understanding Spaghetti Models with Sequence Clustering for ProM." In: *Business Process Management Workshops*. Cham: Springer Berlin Heidelberg, 2010, pp. 92–103. DOI: 10.1007/978-3-642-12186-9_10.

[Ver+11] H. M. W. Verbeek, Joos C. A. M. Buijs, Boudewijn F. V. van Dongen, and Wil M. P. van der Aalst. "XES, XESame, and ProM 6." In: *Information Systems Evolution*. Springer Berlin Heidelberg, 2011, pp. 60–75. DOI: 10.1007/978-3-642-17722-4_5.

[VTM08] K. Vergidis, A. Tiwari, and B. Majeed. "Business Process Analysis and Optimization: Beyond Reengineering." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.1 (Jan. 2008), pp. 69–82. DOI: 10.1109/tsmcc.2007.905812.

[Wan+18] Pan Wang, Wen'an Tan, Anqiong Tang, and Kai Hu. "A Novel Trace Clustering Technique Based on Constrained Trace Alignment." In: *Human Centered Computing*. Springer International Publishing, 2018, pp. 53–63. DOI: 10.1007/978-3-319-74521-3_7.

[WRR08] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. "Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems." In: *Data & knowledge engineering* 66.3 (2008), pp. 438–466. DOI: 10.1007/978-3-540-72988-4.

[WBT11] Phil Weber, Behzad Bordbar, and P. Tino. "Real-Time Detection of Process Change using Process Mining." In: *ICCSW* May 2017 (2011), pp. 108–114.

[Wee+12] Jochen De Weerdt, Manu De Backer, Jan Vanthienen, and Bart Baesens. "A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs." In: *Information Systems* 37.7 (Nov. 2012), pp. 654–676. ISSN: 0306-4379. DOI: 10.1016/j.is.2012.02.004.

[Wee+13] Jochen De Weerdt, Seppe vanden Broucke, Jan Vanthienen, and Bart Baesens. "Active Trace Clustering for Improved Process Discovery." In: *IEEE Transactions on Knowledge and Data Engineering* 25.12 (2013), pp. 2708–2720. DOI: 10.1109/tkde.2013.64.

[Wei+11] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, Jan Mendling, and Mathias Weske. "Process compliance analysis based on behavioural profiles." In: *Information Systems* 36.7 (Nov. 2011), pp. 1009–1025. DOI: 10.1016/j.is.2011.04.002.

[WR11] A. J. M. M. Weijters and J. T. S. Ribeiro. "Flexible Heuristics Miner (FHM)." In: *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. Vol. 334. December. IEEE, Apr. 2011, pp. 310–317. DOI: 10.1109/CIDM.2011.5949453.

[WGN13] Michael Werner, Nick Gehrke, and Markus Nüttgens. "Towards Automated Analysis of Business Processes for Financial Audits." In: (Mar. 2013), pp. 375–389.

[WW08] Graham Wills and Leland Wilkinson. "AutoVis: Automatic Visualization." In: *Information Visualization* 9.1 (Dec. 2008), pp. 47–69. DOI: 10.1057/ivs.2008.27.

[WFH11] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, 2011, p. 664. ISBN: 0080890369.

[Won+16] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. "Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations." In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 649–658.

[Won+17] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. "Voyager 2: Augmenting Visual Analysis with Partial View Specifications." In: *ACM Human Factors in Computing Systems (CHI)*. 2017. URL: `http://idl.cs.washington.edu/papers/voyager2`.

[Yan+16] Sen Yang, Xin Dong, Moliang Zhou, Xinyu Li, Shuhong Chen, Rachel Webman, Aleksandra Sarcevic, Ivan Marsic, and Randall S. Burd. "VIT-PLA: Visual Interactive Tool for Process Log Analysis." In: *KDD Workshop on Interactive Data Exploration and Analytics* 5 (2016), pp. 130–137.

[YK15] Onur Yilmaz and Pinar Karagoz. "Generating Performance Improvement Suggestions by using Cross-Organizational Process Mining." In: *6th International Symposium on Data-driven Process Discovery and Analysis (SIMPA15)* 6 (2015), pp. 3–17.

Erklärung

Hiermit erkläre ich, die vorgelegte Arbeit mit dem Titel

Intelligent Computer-assisted Process Mining

selbstständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel erstellt zu haben.

Hiermit erkläre ich weiterhin, dass von mir bisher kein Promotionsversuch unternommen wurde.

Darmstadt, den 06. Mai 2020

Alexander Niklas Seeliger