DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
15

ADRIANO AUGUSTO

Accurate and Efficient Discovery of Process Models from Event Logs

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

15

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

15

ADRIANO AUGUSTO

Accurate and Efficient Discovery of Process Models from Event Logs

UNIVERSITY OF TARTU
Press

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in informatics on February 11, 2020 by the Council of the Institute of Computer Science, University of Tartu.

Supervisor

Prof.	Marlon Dumas University of Tartu, Estonia
Prof.	Marcello La Rosa University of Melbourne, Australia

Opponents

Assoc. Prof.	Benoit Depaire Hasselt University, Belgium
Assoc. Prof.	Remco Dijkman Eindhoven University of Technology, The Netherlands

The public defense will take place on March 23, 2020 at 10:15 at Delta Building, Narva mnt 18, Room 1020.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

Copyright © 2020 by Adriano Augusto

ISSN 2613-5906
ISBN 978-9949-03-290-7 (print)
ISBN 978-9949-03-291-4 (pdf)

University of Tartu Press
`http://www.tyk.ee/`

Ad maiora.

ABSTRACT

Everyday, organizations deliver services and products to their customers by enacting their business processes, the quality and efficiency of which directly influence the customer experience. In competitive business environments, achieving a great customer experience is fundamental to be a successful company. For this reason, companies rely on programmatic business process management in order to discover, analyse, improve, automate, and monitor their business processes.

One of the core activities of business process management is *process discovery*. The goal of process discovery is to generate a graphical representation of a business process, namely business process model, which is then used for analysis and optimization purposes. Traditionally, process discovery has been a time-consuming activity performed either by interviewing relevant process stakeholders and employees, or by observing process participants in action, or analysing process reference documentation. However, with the diffusion of information systems, and specialised software in organizational settings, a new form of process discovery is slowly emerging, which goes by the name of *automated process discovery*.

Automated process discovery allows business analysts to exploit process' execution data (recorded into so-called *event logs*) to automatically generate process models. Discovering high-quality process models is extremely important to reduce the time spent to enhance them and avoid mistakes during process analysis. The quality of a discovered process model depends on both the input data and the automated process discovery approach (APDA) that is applied. In this thesis, we provide a systematic literature review (SLR) and benchmark of the state-of-the-art APDAs. Our SLR analyses 34 APDAs, while our benchmark evaluates six representative APDAs on more than 20 real-life datasets and seven quality measures. Our SLR and benchmark highlight that existing APDAs are affected by one (or more) of the following three limitations: (i) they achieve limited accuracy; (ii) they are computationally inefficient to be used in practice; (iii) they discover syntactically incorrect process models. To address these limitations, we propose a novel APDA, namely *Split Miner*, that we assessed through our benchmark. The results of our evaluation show that Split Miner outperforms the state-of-the-art APDAs over multiple quality dimensions.

Most of the APDAs we assessed in our benchmark, including Split Miner, require a number of input parameters. The quality of the discovered models depends on how these parameters are tuned. We have found that automated hyper-parameters optimization leads to considerable improvements in the quality of the models produced by an APDA (including Split Miner). The quality improvement APDAs achieve via hyper-parameters optimization comes, however, at the cost of longer execution times and higher computational requirements, due to the inefficiency of existing accuracy measures for APDAs (in particular precision) and the lack of efficient solution-space exploration techniques available for APDAs.

This thesis tackles the problem of APDAs optimization in two parts. First, we propose a set of accuracy measures based on Markovian abstractions, and show that our Markovian accuracy measures are faster than existing accuracy measures and fulfil a set of desirable properties that state-of-the-art measures do not. Next, we propose an optimization framework powered by single-solution-based metaheuristics, which employ our Markovian accuracy measures to efficiently explore the solution-space of APDAs based on *directly-follows graphs* (DFGs), in order to discover a process model with the highest accuracy. The evaluation of our optimization framework highlights its effectiveness in optimizing DFG-based APDAs, showing that it allows APDAs to explore their solution-space beyond the boundaries of hyper-parameter optimization and most of the times in a faster manner, ultimately discovering more accurate process models in less time, compared to hyper-parameters optimization.

In order to foster reproducibility and reuse, all the artifacts designed and developed for this thesis are publicly available as open-source Java command-line applications. Split Miner, our core contribution to the field of automated process discovery, has also been integrated into Apromore, an open-source business process analytics platform used by academics and practitioners worldwide.

CONTENTS

1. Introduction 15

1.1. Research Area 15
1.2. Research Problem 18
1.3. Research Method 20
1.4. Contributions to the Research Area 22
1.5. Thesis Structure 23

2. Background 24

2.1. Business Processes 24
2.2. Business Process Modelling Languages 26
    2.2.1. Business Process Model and Notation (BPMN) 26
    2.2.2. Petri Nets 33
    2.2.3. Other Process Modelling Languages 34
2.3. Process Discovery 36

3. State of the Art 39

3.1. Automated Process Discovery 39
3.2. Methodology 40
    3.2.1. SLR Research Questions 40
    3.2.2. Search String Design 41
    3.2.3. Study Selection 42
3.3. Classification of the Studies 43
    3.3.1. Model Type and Language (RQ1.2) 44
    3.3.2. Procedural Language Constructs (RQ1.3) 49
    3.3.3. Implementation (RQ1.4) 49
    3.3.4. Evaluation Data and Domains (RQ1.5) 49
3.4. Threats to Validity 50
3.5. Summary 51

4. Benchmark 53

4.1. APDAs Selection 53
4.2. Evaluation Measures 55
4.3. Setup and Datasets 57
4.4. Benchmark Results 59
4.5. Threats to Validity 68
4.6. Related Work 70
4.7. Summary 70

5. Split Miner 72
5.1. Approach 72
5.1.1. Directly-Follows Graph and Short-Loops Discovery 73
5.1.2. Concurrency Discovery 75
5.1.3. Filtering 75
5.1.4. Filtered PDFG to BPMN Process Model 81
5.1.5. Splits Discovery 81
5.1.6. Joins Discovery 88
5.1.7. OR-joins Minimization 90
5.1.8. Time Complexity 94
5.2. Semantic Properties of the Discovered Model 94
5.2.1. Preliminaries 95
5.2.2. Proofs 96
5.3. Evaluation 98
5.3.1. Evaluation Results 99
5.4. Summary 109

6. Markovian Accuracy 111
6.1. Fitness and Precision in Automated Process Discovery 112
6.1.1. Fitness Measures 112
6.1.2. Precision Measures 113
6.1.3. Fitness Propositions 115
6.1.4. Precision Propositions 116
6.2. K^{th}-order Markovian Abstraction 118
6.2.1. Generating the M^{k}-abstraction of an Event Log 120
6.2.2. Generating the M^{k}-abstraction of a Process 122
6.3. Comparing Markovian Abstractions 123
6.3.1. Markovian Abstraction-based Fitness 125
6.3.2. Proofs of the 7-Propositions of Fitness 125
6.3.3. Markovian Abstraction-based Precision 127
6.3.4. Proofs of the 8-Propositions of Precision 128
6.4. Evaluation 132
6.4.1. Qualitative Evaluation Dataset 132
6.4.2. Qualitative Evaluation of MAF^k 134
6.4.3. Qualitative Evaluation of MAP^k 135
6.4.4. Quantitative Evaluation Setup 139
6.4.5. Quantitative Evaluation of MAF^k 139
6.4.6. Quantitative Evaluation of MAP^k 140
6.4.7. The Role of k 146
6.5. Summary 147

7. Optimized Automated Process Discovery **149**
7.1. Optimization Metaheuristics 150
7.2. Metaheuristic Optimization Framework 152
7.2.1. Preliminaries 152
7.2.2. Approach Overview 153
7.2.3. Adaptation of the Optimization Metaheuristics 153
7.2.4. Framework Instantiation 156
7.3. Evaluation 159
7.3.1. Dataset and Experimental Setup 159
7.3.2. Results 160
7.3.3. Statistical Analysis 168
7.3.4. Discussion 169
7.4. Summary 170

8. Conclusion **171**
8.1. Summary of Contributions 171
8.2. Future Work 172

Bibliography **174**

Acknowledgement **189**

Summary in Estonian **190**

Curriculum Vitae **192**

Elulookirjeldus (Curriculum Vitae in Estonian) **193**

List of original publications **194**

LIST OF FIGURES

1. Business Process Management lifecycle [1] 16
2. Example of a process model in BPMN notation. 17
3. Amazon *order-to-cash* business process (customer perception). . . 27
4. Examples of events. 28
5. Examples of activities. 28
6. Collapsed and expanded subprocess (resp. above and below). . . . 30
7. Samples of gateways. 31
8. Example of proper and improper use of XOR gateways. 31
9. Example of proper and improper use of AND gateways. 32
10. Example of proper and improper use of OR gateways. 32
11. Example of Petri net elements, and firing transition. 34
12. Amazon *order-to-cash* business process (customer perception), as a Petri net. 35
13. Number of studies over time. 43
14. Overview of the proposed approach. 73
15. Processing of the directly-follows graph. 74
16. Best incoming and outgoing edges selected by Algorithm 2 and 3 from the PDFG. 80
17. Processing of the BPMN model. 81
18. Splits discovery example. 86
19. Joins discovery examples. 89
20. OR-joins minimization example. 90
21. OR-joins minimization algorithm. 92
22. Examples of processes in the BPMN language. 119
23. From left to right: the M^1-abstraction of the Flower Process, Process-Y, Process-X and the event log L^*. 120
24. From left to right, the M^2-abstraction of the Flower Process, Process-Y, Process-X and the event log L^*. 120
25. Qualitative evaluation artificial models as in [2]. 133
26. Model discovered by IM from the SEPSIS log. 146
27. Model discovered by SM from the SEPSIS log. 146
28. Overview of our approach. 153
29. BPIC14_f models discovered with SIMA_sm (above) and SM (below). 163
30. RTFMP models discovered with SIMA_sm (above) and SM (below). 163

LIST OF TABLES

1. Example of an event log. 17
2. APDAs assessed by De Weerdt et al. [3] (left) and the respective successors (right). 42
3. Overview of the 34 primary studies resulting from the search (ordered by year and author). 52
4. Descriptive statistics of public logs. 58
5. Descriptive statistics of proprietary logs. 59
6. Default parameters evaluation results for the BPIC logs. 60
7. Default parameters evaluation results for the public logs. 61
8. Default parameters evaluation results for the proprietary logs - Part 1/2. 61
9. Default parameters evaluation results for the proprietary logs - Part 2/2. 62
10. Best score frequencies for each quality dimension (default parameters evaluation). 63
11. Scores of the models with the best F-score discovered with hyper-parameter optimization (public logs). 65
12. Scores of the models with the best F-score discovered with hyper-parameter optimization (proprietary logs). 66
13. Best scores achieved in hyper-parameter evaluation by each approach on each quality dimension (public logs). 67
14. Best scores achieved in hyper-parameter evaluation by each approach on each quality dimension (proprietary logs). 67
15. Filtering algorithm example. 80
16. Splits discovery example. 87
17. Split Miner default parameters evaluation results for the public logs – Part 1/2. 99
18. Split Miner default parameters evaluation results for the public logs – Part 2/2. 100
19. Split Miner default parameters evaluation results for the proprietary logs - Part 1/2. 101
20. Split Miner default parameters evaluation results for the proprietary logs - Part 2/2. 102
21. Best score frequencies for each quality dimension (default parameters evaluation, inc. Split Miner). 103
22. Scores of the models with the best F-score discovered with hyper-parameter optimization (public logs), inc. Split Miner. 105
23. Scores of the models with the best F-score discovered with hyper-parameter optimization (proprietary logs), inc. Split Miner. 106
24. Best scores achieved in hyper-parameter evaluation by each approach on each quality dimension (public logs), inc. Split Miner. 107

25. Best scores achieved in hyper-parameter evaluation by each approach on each quality dimension (proprietary logs), inc. Split Miner. . . 108
26. Fitness propositions fulfiled by existing fitness measures (according to [4]). 116
27. Precision propositions fulfiled by existing precision measures (according to [4]), including the Axioms 1-5 (A1-A5) as discussed in [5]. 117
28. Log L^*. 119
29. Precision propositions [4] and axioms [5] fulfiled by MAP^k, and for which value of k. 128
30. Test log [2]. 133
31. Values of the fitness measures over the synthetic dataset. 135
32. Model ranking induced by the fitness measures over the synthetic dataset. 136
33. Values of the precision measures over the synthetic dataset. 137
34. Model rankings induced by the precision measures over the synthetic dataset. 137
35. Comparison of fitness measures over the 20 real-life logs. 141
36. Models ranking yielded by fitness measures over the 20 real-life logs. 142
37. Time performance (in seconds) of fitness measures using the twelve public logs. 143
38. Time performance (in seconds) of fitness measures using the eight proprietary logs. 143
39. Comparison of precision measures over 20 real-life logs. 144
40. Models ranking yielded by precision measures over 20 real-life logs. 145
41. Time performance (in seconds) of precision measures using the twelve public logs ('+' indicates a result obtained on a subset of the twelve logs, due to some of the measurements not being available). 146
42. Time performance (in seconds) of precision measures using the eight proprietary logs. 146
43. Comparative evaluation results for the public logs - Split Miner. . . 161
44. Comparative evaluation results for the proprietary logs - Split Miner. 162
45. Comparative evaluation results for the public logs - Fodina. 165
46. Comparative evaluation results for the proprietary logs - Fodina. . 166
47. Comparative evaluation results for the public and proprietary logs - Inductive Miner. 167
48. Summary of the Mann-Whitney U-tests [6]. 169

LIST OF ABBREVIATIONS

Automated Process Discovery Approach	APDA
Business Process Management	BPM
Business Process Model and Notation	BPMN
Directly-follows Graph	DFG
Evolutionary Tree Miner	ETM
Fodina Miner	FO
Heuristics Miner	HM
Hyper-Parameter Optimization	HPO
Inductive Miner	IM_d
Inductive Miner Infrequent	IM
Information System	IS
Markovian Abstraction-based Fitness	MAF
Markovian Abstraction-based Precision	MAP
Process Modelling Guidelines	PMG
Research Question	RQ
Single-entry Single-exit	SESE
Structured Heuristics Miner	SHM (S-HM_6)
Split Miner	SM
Systematic Literature Review	SLR

1. INTRODUCTION

1.1. Research Area

Everyday, people take part in one or more processes even simultaneously: sometimes as resources, sometimes as customers, sometimes as data. Often, people do not notice it, for they do not realize they are acting within a process or they are just not interested in insights about the process they are temporarily involved in. Differently, companies and industries are growing their interest in studying and understanding their processes, since these the soul of their business. Indeed, quality of services, quantity of revenue, amount of costs, customers satisfaction, and ultimately the overall company success depend on a company's business processes. In this context, the discipline of Business Process Management (BPM) was born with the aim of helping companies to analyse, assess and monitor their business processes and improve them, regardless of the type of organization, its provided services, characteristics and targets [1]. Figure 1 shows the BPM lifecycle [1]: the diagram formally explains how business stakeholders and information technology (IT) specialists collaborate through each distinct phase to achieve business improvement. Here, we give a brief description of each phase.

- **process identification**: it is necessary to produce a process architecture, which captures all the processes of the organization and their relationships.
- **process discovery**: a business process is selected from the process architecture and its model is discovered (either manually or automatically). A process model is a graphical representation of the business process, which should clearly capture its workflow. The representation of the process at this stage is called *as-is process model.*
- **process analysis**: the IT specialists analyse the *as-is process model* using a set of analysis techniques. The outcome of this analysis is a list of issues and weaknesses of the process, as well as an assessment of their impact on process performance
- **process redesign**: to solve the identified issues and weaknesses, possible changes are evaluated. Successively, the chosen changes are applied to the *as-is process model.* The process is redesigned and a new model is drawn, namely the *to-be process model.*
- **process implementation**: the *to-be process model* is put into effect in the organization workflow. This can entail: the changing of the process structure (e.g. swapping resources, splitting or merging activities, adding parallel activities etc.); the adoption of process automation; the deployment (or upgrade) of IT systems to monitor the effectiveness of the changes.
- **process monitoring and controlling**: the process is monitored, and its data is continuously collected, typically using IT systems. IT specialists can perform on-line and off-line analysis of the new process, for assessing the

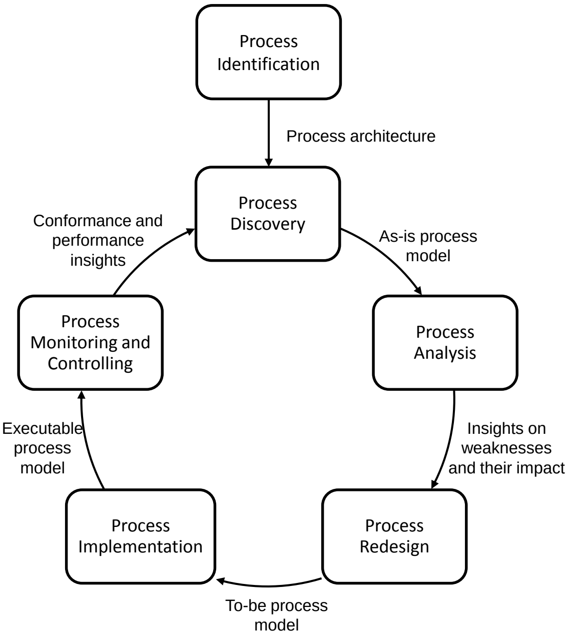


Figure 1: Business Process Management lifecycle [1]

adherence of the enacted process to the new process design. Any deviation may give rise to another execution of the BPM lifecycle.

Our research project sits on the second phase of the BPM lifecycle: *process discovery*. Process discovery rapidly evolved over the last two decades, moving from manual methods such as employees' interviews, workshops, observations, and documents analysis, to automated process discovery from *event logs*. However, automated process discovery is not yet an established practice, as this field of research has been experiencing a tremendous evolution in the last decade.

Modern IT systems allow companies to easily collect detailed information about their business processes, including records of their execution events, such as the creation of a case or the execution of an activity within an ongoing case. These records are known as *event logs*. We can picture an event log as a file recording what happened and how within a company (e.g. activities performed by employees, their chronological order, their execution times, their associated data), in other words, an event log contains a detailed history of process executions. As an example, Figure 2 and Table 1 show (respectively) a process model (in the BPMN language) and an event log recorded during its execution. Given the huge amount of data stored into event logs, the most efficient way to exploit its concealed information is via the application of automated techniques. In this setting, the discipline of *Process Mining* was born to provide data-driven automated support along various phases of the BPM lifecycle, especially those phases where the data analysis plays a central role. Process mining encompasses techniques that can be grouped into two major families: *operational techniques* and *tactical tech-*

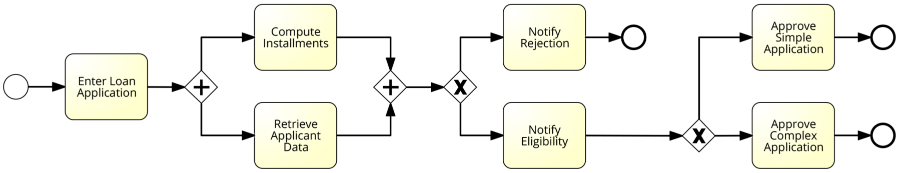


Figure 2: Example of a process model in BPMN notation.

niques. The former family groups techniques whose goal is to generate insights in real-time during the process execution, such as estimating the remaining process execution time; or the probability of a negative event to happen; or the likelihood of a specific process outcome. The latter family groups techniques whose goal is to help analysts to discover, analyse, and periodically monitor the process execution in order to understand how the process is performed, what are its weaknesses, and how the process can be improved.

Case ID	**Event Name**	**Timestamp**	⋯
01	Enter Loan Application	2007-11-09 T 11:20:10	⋯
01	Compute Instalments	2007-11-09 T 11:25:15	⋯
02	Enter Loan Application	2007-11-09 T 11:30:40	⋯
01	Retrieve Applicant Data	2007-11-09 T 11:40:50	⋯
02	Retrieve Applicant Data	2007-11-09 T 11:50:00	⋯
02	Compute Instalments	2007-11-09 T 12:00:30	⋯
02	Notify Eligibility	2007-11-09 T 12:10:45	⋯
03	Enter Loan Application	2007-11-09 T 13:23:15	⋯
03	Compute Instalments	2007-11-09 T 13:30:35	⋯
01	Notify Rejection	2007-11-09 T 14:45:00	⋯
02	Approve Simple Application	2007-11-09 T 15:00:30	⋯
⋯	⋯	⋯	⋯

Table 1: Example of an event log.

This thesis focuses on tactical process mining techniques, which can be classified into four categories [1, 7]

- **process discovery** techniques, address the problem of *automated process discovery*. Techniques belonging to this category take as input an event log and output a process model whose behaviour closely matches the behaviour observed in the event log.
- **conformance checking** techniques, aim to compare process models against event logs in order to identify mismatches and highlight behavioural differences. For example, a process model might tell us that after checking the plausibility of a loan application, we must check the credit history, whilst in the event log, after the plausibility check, we sometimes do not observe the credit history check. Behavioural mismatches between the process model and the event log can be due to an error or, more often than not, due to an

exception that is not captured in the process model. In general, techniques belonging to this family are helpful during the process analysis phase of the BPM lifle-cycle, and can also be used to assess the quality of automated process discovery techniques.

- **performance mining** techniques, which take as input a process model (either designed manually or automatically discovered) and its event log and output a version of the process model enriched with performance statistics. For example, these can be mean execution times, waiting times, costs, defects rates.
- **variants analysis** techniques, aim to compare two or more variants of the same business process, via their process executions, in order to understand and/or analyse any execution difference (and its root causes). These techniques take as input two or more event logs (corresponding to the process variants) and output a description of the differences between the event logs. For example, given two event logs, the first may contain all the process executions where the customer was satisfied, while the second one may contain all the executions leading to customer complaints. In this scenario, variants analysis techniques allow us to understand what is done differently in those cases where the customer complains, in order to understand why such complaints arise in the first place.

This thesis focuses on automated process discovery techniques, as well as conformance checking techniques insofar as the latter are used to evaluate the quality of the process models discovered via automated process discovery.

1.2. Research Problem

In the past two decades, many research studies addressed the problem of automated discovery of business process models from event logs [8–12]. Despite the richness of proposals, the state-of-the-art automated process discovery approaches (APDAs) struggle to *systematically* discover accurate process models. When the input event log is simple, some state-of-the-art APDAs can output accurate and simple process models, however, as the complexity of the input data increases, the quality of the discovered process models can worsen quickly. Given that the majority of the real-life event logs highly complex (i.e. containing noise and incomplete data), state-of-the-art APDAs turn to be unreliable and not purposeful. More specifically, they suffer from at least one of the following three limitations when receiving as input large and complex event logs [13] (i) they achieve limited accuracy (i.e. low scores of fitness and/or precision); (ii) they are computationally inefficient to be used in practice; (iii) they discover syntactically incorrect process models.

Automatically discovered process models (and in turn APDAs) can be assessed via four quality dimension [7]:

- **Fitness** (a.k.a. recall) estimates the amount of behaviour observed in the event log that can be found in the process model.
- **Precision** estimates the amount of behaviour captured in the process model that can be found in the event log.
- **Generalization** assesses to what extent the process model captures behaviour that, despite not being present in the event log, can eventually be produced by the (original) process under observation.
- **Complexity** assesses the structure and understandability of the process model via several metrics (e.g. size, control flow complexity, model structuredness) [14].

Along these four quality dimensions, a fifth property ensures the semantic correctness of the process model, namely, *soundness*. The notion of *soundness* has been defined on Workflow nets [15] as a proxy for assessing their syntactic correctness, but can be adapted to BPMN process models as follows. A BPMN process model with one start and one end event is sound if and only if the following three properties hold altogether: (i) any arbitrary process execution can reach the end event (option to complete); (ii) no end events can be triggered more than once during a process execution (proper completion); (iii) for each activity of the process, there exists at least one process execution that triggers the activity (no dead activities).

Achieving in a robust and scalable manner the best trade-off between the four quality dimensions, while ensuring soundness, has proved elusive. In particular, we can identify two kinds of APDAs: those focusing more on the *simplicity*, the *soundness* and either the *precision* [10] or the *generalization* [9] of the discovered process model, and those focusing more on its *fitness* and its *precision* at the cost of *simplicity* and/or *soundness* [8, 11, 12].

The goal of our research project is to investigate the state-of-the-art APDAs, identify and assess their strengths and limitations, and design a novel effective and efficient APDA such that: it overtakes the limitations of the existing APDAs, and sums their strengths up to solve the problem of automated process discovery.

In line with our goal, we formulated the following research questions:

RQ1. What are the state-of-the-art automated process discovery approaches, their strengths and limitations?

RQ2. How to strike a trade-off between the various quality dimensions in automated process discovery in an *effective* and *efficient* manner?

RQ3. How can the accuracy of an automated process discovery approach be efficiently optimized?

Whilst *RQ2* synthesises the main research problem and the main purpose of our research project, answering *RQ1* is a fundamental preliminary step to investigate (and ultimately answer) *RQ2*. Lastly, *RQ3* is necessary to understand whether the outcome of *RQ2* leaves (or does not) space for further improvements, and in a positive case how to efficiently achieve such improvements.

To answer *RQ1* an extensive literature review is required. A consistent, coherent, and reproducible benchmark must be designed to asses the state-of-the-art APDAs thoroughly, fairly, and equally. The results of the benchmark will allow us to identify the strengths and the weaknesses of the state-of-the-art APDAs. The outcome of *RQ1* is the starting point for setting the quality criteria of the solution for our main research problem, summarised in *RQ2*. In the latter, we use the term *effective* to highlight that an APDA should output as *accurate* and *simple* as possible process models (regardless of the input data) within *acceptable* execution time, i.e. *efficiently*. We can synthesise into three criteria the quality of the APDA we will search for in our research project:

- *Accurate*: the discovered process models should score high values of fitness and precision, such that their F-score is equal or greater than 80% on average, since this a difficult milestone for the state-of-the-art APDAs.
- *Simple*: the discovered process models should be small in size and exhibit low control flow complexity, with respect to (w.r.t.) process models discovered by state-of-the-art APDAs.
- *Efficient*: the APDA should discover process models in real time, i.e. within one second.

1.3. Research Method

Given that our research area within the field of Information Systems (IS), to answer our research questions and achieve our research goal, we followed the Design Science in IS research method [16]. Hevner et al. [16] propose the following seven guidelines (G1-G7) to address a research problem in the IS field.

G1. *Design as an Artifact.* The ultimate outcome of the research project should be one (or more) *novel* and *purposeful* artifact(s).

G2. *Problem Relevance.* As, in G1, the term *purposeful* highlights, the artifacts produced as outcome of the research project should address (and solve) one (or more) *relevant* problem(s) in the research area. The term *relevant* means that the problem has not been addressed before or the proposed solutions available in the state of the art leave space for improvements.

G3. *Design Evaluation.* Each of the produced artifacts should be evaluated appropriately, according to their type. The evaluation results should prove the quality, the utility, and the efficacy of the designed artifacts.

G4. *Research Contributions.* As, in G1, the term *novel* highlights, the produced artifacts should provide an innovative solution for the addressed research problem. The innovation can be of two types: (i) the artifacts solve an unsolved problem; or (ii) the artifacts solve more efficiently and/or more effectively a problem already solved.

G5. *Research Rigor.* All the produced artifacts should be rigorously defined in a formal way, to ensure reproducibility, consistency, and coherence.

G6. *Design as a Search Process.* Each of the artifacts should be designed applying a search process. A problem space should be defined and a set of solution criteria identified. Once a possible solution is designed, its quality should be verified using the criteria initially set, and the solution should be improved if necessary. This iterative flow is fundamental to reach an optimal or quasi-optimal solution to the problem.

G7. *Communication of Research.* All the results and outcomes of the research project should be delivered/presented to both academics and practitioners. During our research project, we published three conference papers and two journal articles, whilst two more journal articles are currently under review.

In this thesis, we implemented the above guidelines as follows. (G2) To ensure our work would focus on relevant research problems and we could deliver novel purposeful artifacts, we began our project by reviewing and analysing the studies related to our research area. Given that the latest literature review of automated process discovery was dated 2012 [3], we performed a systematic literature review to cover also the recent years. The insights derived from such a preliminary work allowed us to identify four research gaps/problems, precisely: (i) the lack of a documented and reproducible benchmark for APDAs; (ii) three major limitations affecting the state-of-the-art APDAs; (iii) the low efficiency of accuracy measures for assessing the quality of automatically discovered process models; and (iv) the lack of a general optimization method for APDAs. (G1) Accordingly, we set to deliver four novel artifacts to our research community: (i) a modular benchmark for assessing APDAs; (ii) an APDA that would overcome the current limitations; (iii) a family of efficient accuracy measures; and (iv) an optimization framework for APDAs. (G5) We ensured that each of the artifacts we designed as part of this thesis is formally described. Where algorithms were designed, we rigorously justified all the design choices, we reported the pseudo-code and discussed it in a procedural manner to guarantee transparency and understandability. Furthermore, we meticulously described the setup of all the evaluations carried-on to assess our artifacts and released the source code and data used in our experiments, so that they could be easily reproduced. (G3-G4) We assessed all the artifacts we produced by performing one or more empirical evaluations based on large real-life datasets and where necessary employing also artificial datasets. Our evaluations allowed us to analyse the quality and properties of our artifacts, highlighting their ability to solve the identified research problems more efficiently and/or more effectively than available baselines. (G6) Finally, the artifacts we introduce in this thesis are the outcomes of a long iterative design flow, which ultimately led to their final (optimal) versions. This is clearly documented for three out of the four artifacts, whose earlier versions have reached the research community as conference proceedings, before the latest version was published in a journal. (G7) In fact, all our research work has been presented to conferences and published in journals. The rank of the conferences and journals that accepted our research work ensure that our results and artifacts will be at reach of both the research community and the practitioners in the area of BPM and process mining. Furthermore, our novel APDA, Split Miner, was integrated into *Apromore*: the open-source web-based platform for business process analytics developed by the BPM Research Team at the University of Melbourne. [1] Apromore is known to practitioners and academics, and it was recently added to the *Gartner's Market Guide for Process Mining*.

1.4. Contributions to the Research Area

This thesis presents five major contributions to the identified research area of automated process discovery. Our first major contribution is a systematic literature review (SLR) of the state-of-the-art APDAs [13]. Our SLR covers more than 80 research studies addressing the problem of automated process discovery, published between 2012 and 2017, and it reports on the features characterizing more than 30 existing APDAs. To assess the performance of the existing APDAs, and analyse their strengths and weaknesses, we designed and implemented a benchmark framework to evaluate in an automated manner the existing APDAs. The design of our benchmark allows academics and practitioners to extend the automated evaluation including new APDAs and/or new type of measurements [13]. The benchmark framework represents our second major contribution, which combined with our SLR allowed us to answer RQ1. The insights gained by answering RQ1 helped us to design a novel APDA, namely Split Miner [17, 18], that satisfies the quality criteria we described in Section 1.2. The algorithms Split Miner stands on address our RQ2, whilst Split Miner (as an artifact) is our third and most important contribution to the research area.

Similarly to the vast majority of the existing APDAs, the accuracy of Split Miner can be fine-tuned via its input parameters. To perform such a tuning, it is necessary to rely on techniques that measure the accuracy of the discovered process models during the tuning process. Unfortunately, such techniques are inefficient and ultimately lead to inefficient parameters tuning. Moreover, existing conformance checking techniques do not fulfil basic formal properties that one would expect from accuracy measures. To address this shortcoming, we propose our fourth contribution: the Markovian accuracy, a novel technique for approximate fast computing of process model accuracy with respect to an event log [19,20]. Employing our Markovian accuracy, we explore several optimization metaheuristics to fine-tune the accuracy of the process models discovered by the most reliable APDAs (including Split Miner), at the expense of the execution time. This leads to our fifth contribution: an optimization framework for APDAs, powered by metaheuristics optimization adapted to the context of automated process discovery [21].

---

[1] `https://apromore.org/`

1.5. Thesis Structure

The rest of this thesis is organized as follows. Chapter 2 provides an overview of business process modelling and the available modelling languages, and we formally introduce the problem of process discovery. Chapter 3 reports an extensive literature review in automated process discovery. Chapter 4 presents our benchmark for assessing the state-of-the-art APDAs, shows the results of the evaluation, and highlights the strengths and weaknesses of the assessed APDAs. Chapter 5 formally describes our novel APDA, Split Miner, focusing on its properties and its performance when compared to the state-of-the-art APDAs. Chapter 6 introduces two new methods to compute fitness and precision, which allow us to assess the accuracy of automatically discovered process models in a fast, yet approximate, manner. Chapter 7 proposes a novel optimization framework based on metaheuristics to optimize the F-score of fitness and precision of a specific class of APDAs, which includes Split Miner. Chapter 8 draws the conclusions of the thesis, summarising the core contributions and providing future work avenues.

2. BACKGROUND

In this chapter, we draw the starting line of our research project. First, in Section 2.1, we introduce the concept of *business process* with the help of a simple real-life case. In Section 2.2, we explain why it is important to represent business processes through models, and what are the most popular modelling languages to capture business processes, giving an overview of their graphical elements and semantics. Lastly, in Section 2.3, we introduce the problem of process discovery.

2.1. Business Processes

“Business processes are what companies do whenever they deliver a service or a product to customers.” [1]

To understand the meaning of this sentence, let us consider a simplified view of the the process for handling orders (also known as an order-to-cash process) at a popular online retailer, namely Amazon. [1] [2] The process starts when an Amazon customer, after having filled the digital cart with a certain number of items, wants to checkout. At the click of the button *checkout* the process is triggered, so that we can identify the *checkout request* as the starting event of the process. What follows next is a sequence of well defined activities: the user is asked for the shipping address; for the shipping type (standard, express, etc.); and for the payment method and details. Then, Amazon checks the validity of the payment details, and (i) if everything is correct and valid, Amazon sends an order confirmation email; otherwise (ii) Amazon rejects the order, ending the process at this point with the event *order aborted*. Assuming the order is not rejected, after the confirmation email, the items are prepared for the shipping, and when ready the payment is charged. If the payment is unsuccessful, the order would be aborted as in the case of wrong payments details. Instead, if the payment is successful the items are shipped, and the receipt is sent to the user via email. In the latter case, the process would reach the end event *order fulfilled*, and the customer would soon receive the purchased items.

This simple example of *order-to-cash* process shows the activities between the customer checkout request and the fulfilment of the customer order. The name *order-to-cash* summarizes the input and the output of the process, which are (respectively) the customer order and the money Amazon receives for fulfilling it. Whilst the service and the product(s) delivered by Amazon are (respectively) the e-shopping experience and the items purchased by the customer.

Other examples of common business processes are the following [1]:

---

[1] The process description is simplified. It only considers the viewpoint of the customer (and not the numerous activities that take place inside the company) nor does it consider any rework that occurs when there is a defect in the execution of the process.

[2] The model is a simplified version of the customer perception of the process, e.g. no rework is performed in the case something goes wrong.

- **Quote-to-Order**. This process is frequent in all the companies that sell products or services. It is triggered by the request of a third-party for buying or acquiring a product or a service. The supplier provides a quote to the other party, who can then accept or reject it (in the simplest scenario). When the quote is accepted, an order is generated and what follows next is the *order-to-cash* process. The sequence of *quote-to-order* and *order-to-cash* processes is also known as *quote-to-cash* process.
- **Issue-to-Resolution**. This business process is common within the companies providing products warranty and/or customer support. It starts when a customer requests assistance because he is experiencing an issue with a previously purchased item or he needs support for accessing a service. The process ends when (hopefully) the issue is solved.
- **Application-to-Approval**. Everyone took part to this process at least once in his life, indeed, it is the classic scenario of a customer applying for receiving a service or product. It may be the application for enrolling in a higher education institution; the application for receiving a home loan; or the application for renting an apartment. The process may end either with the approval or rejection of the submitted application. The latter case verifies (usually) when some mandatory criteria for granting the acceptance of the application are not met.
- **Procure-to-Pay**. This is another business process present in most of the companies as well as public organizations.. It is triggered when an employee needs to purchase a product or a service from a third-party supplier. It follows the request of a quote from the procurement department to any available supplier. Then, when one or more quotes are received, the best is selected and the order placed. The process ends when the product or service is (paid and) delivered to the employee who requested it. For each *procure-to-pay* process, usually a *quote-to-cash* process executes within the supplier company.

Given the above examples, we can define a business process as follows.

Definition 2.1.1 (Business Process)**.** *A business process is a sequence of activities performed within an organization to deliver value to the customer, who can be either an internal or external entity. The sequence of activities is triggered by a specific event (bound to a certain input) and leads to a different specific event (bound to a certain output), respectively start event and end event.*

For most of the business processes the identification of the input and the output is straightforward, but, unfortunately, what happens in between is not easy to determine. Indeed, depending on the company, the sequence of activities leading from the start event to the end event of a specific business process (e.g. order-to-cash) may drastically vary, and along with it the quality of the product and/or the service delivered by the company. This explains why certain companies are more successful than their competitors, even providing the same type of products and/or services to their customers, and standing on the same type of business processes.

Business Process Management (BPM) allows a company to: identify their processes; discover how they execute; analyse their quality; improve them; implement the improvements; and finally monitor the processes performance. These six activities compose the BPM lifecycle we introduced in Chapter 1 (Figure 1). However, to discover how a process executes, to analyse its quality, to improve it, to implement it, and even to monitor its performance, it is necessary to have a description of the process either in natural language or graphical representation. Being the latter far more compact and unambiguous,it is most of the times preferred over the former. Consequently, business process models have become valuable artifacts for the companies willing to check and (possibly) improve the quality of their business processes applying the BPM lifecycle.

2.2. Business Process Modelling Languages

In order to perform any sort of analysis of a business process, it is necessary to start from either its natural language description, its execution data (i.e. the process event log), or its graphical representation. Usually, the last type of representation is preferred over the previous two for it is simpler and unambiguous. Whilst the former feature is not a guarantee, but rather one of the objectives of process modelling, the latter is always ensured when the process model is drawn using a standardized process modelling language, which relies on a formally defined set of syntax and semantic rules.

In the following, we introduce the two most popular and widely accepted process modelling languages, precisely: *Business Process Model and Notation* (BPMN) [22]; and *Petri nets* [23–25]. The former is mostly used by practitioners in industry, whilst the latter is widely adopted by academics in their research studies because of its simple and well-defined semantics.

It is important to mention that several other modelling languages are available to graphically represent a process, and depending on the process characteristics and on the type of analysis to perform, some of them could be preferred over the others. Indeed, different modelling languages have different expressiveness power, which ultimately affects the difficulty of the modelling phase and the simplicity of the final model. However, concerning our research project, BPMN and Petri nets are the only two modelling languages that we refer to.

2.2.1. Business Process Model and Notation (BPMN)

BPMN is a modelling language roughly based on classical flowcharts. Its core graphical elements are: events, activities, gateways, and sequence flows. The semantic power of BPMN allows the modelling of the vast majority of business process executions, including complex control flows (i.e. concurrency, inclusive choice, and exclusive choice), data flows, and resources interactions. Figure 3 shows the BPMN representation of the process discussed in Section 2.1. For the

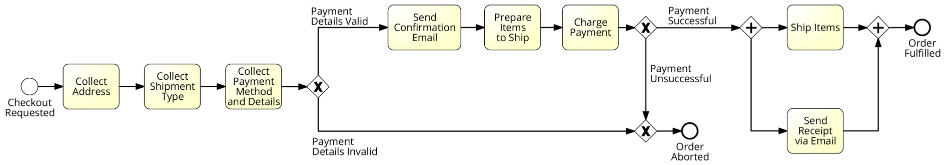


Figure 3: Amazon *order-to-cash* business process (customer perception).

purpose of our project, in this section, we introduce only the BPMN core elements, leaving the extended set of BPMN elements and further details to the BPMN 2.0 specifications [22].

Events. An event represents an expected (or unexpected) happening during a process execution. Events are modelled as circles, and they are characterised by *type*, *direction*, and *trigger*. The type and the trigger of an event are defined by the border of the circle and the inner symbol, figure 4 shows some common events in BPMN.

Depending on when an event materialises during the execution of a process, we identify three types of events:

- *Start event*, which triggers the process execution;
- *Intermediate event*, which happens during the process execution;
- *End event*, which ends the process execution.

Depending on where an event originates, one of the following two directions is assigned to the event:

- *catching*, for events whose source is outside the organisation (e.g. a message is received from a customer);
- *throwing*, for events whose source is inside the organisation (e.g. a confirmation email is sent to a customer).

Start events are only *catching* events, whilst end events are only *throwing* events.

Finally, the trigger of an event is identified by a symbol capturing the cause of the event. The most common triggers are:

- *Message*, it is used to represent the receiving (catching) or the sending (throwing) of a message, its marker is an envelope;
- *Timer*, it is used to synchronise the process execution with an exact date (time), once the date (time) is reached (elapsed) the event triggers. This trigger is for *catching* usage only, and its marker is a clock;
- *Error*, it is used to model an error occurred while executing a process, its usage can be either *catching* or *throwing* depending on the context, and its marker is a thunderbolt;
- *Terminate*, it is used only for end events, it represents the immediate terminations of all the activities still executing within the process; its marker is a black bold circle;
- *None*, it is used to represent a non-specific event, when the information/data is insufficient to determine the real cause of the event. It is often used for

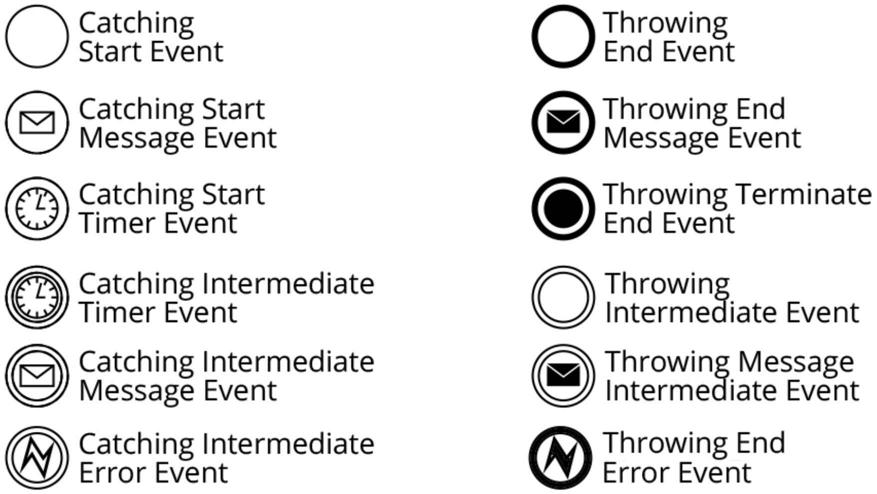

Catching Start Event
Catching Start Message Event
Catching Start Timer Event
Catching Intermediate Timer Event
Catching Intermediate Message Event
Catching Intermediate Error Event
Throwing End Event
Throwing End Message Event
Throwing Terminate End Event
Throwing Intermediate Event
Throwing Message Intermediate Event
Throwing End Error Event

Figure 4: Examples of events.

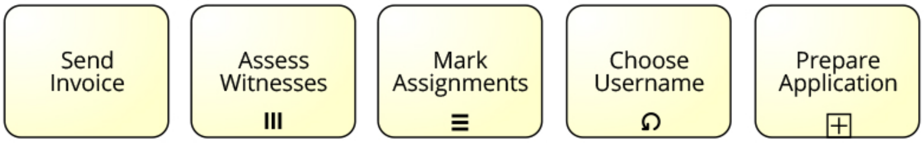


Figure 5: Examples of activities.

start and end events, especially in the context of automated process discovery. It has no marker.

Type, direction, and trigger of an event define its semantics. When the trigger of the event materialises, the event is triggered and immediately consumed, whilst the process execution flow either starts, continues, or ends (depending on the event type).

Activities. An activity allows us to represent a task executed within the process by a resource, this latter can be either a human being, a department, or an information system. To model an activity, we use a labelled box (with rounded corners). The label describes the task's goal, and it usually adheres to the naming convention: an imperative verb followed by a noun, e.g. send invoice. Special activity markers define the type of the activity, and provide further details about the activity execution. We distinguish the following types of activity (depicted in Figure 5, from left to right):

- *Simple Task*, it is used to capture an activity that is intended to be performed as a single unit of work, by a single participant in the process (also called *resource*). We note that the term *activity* may refer either to a (simple) task (i.e. an atomic unit of work) or to an entire subprocess, consisting of multiple tasks.

- *Parallel Multi-Instance*, this type of activity captures the scenario where a given task (or subprocess) needs to be executed in multiple "copies" (called *instances*). Conceptually, this type of activity takes as input a collection of objects (e.g. a collection of line items in a purchase order), and it is executed once for each object in the input collection (i.e. once for each line item in the purchase order). The instances of the activity spawned in this way are executed *concurrently*. Figure 5 shows as an example the assessment of witnesses, the task can be performed in parallel on different witnesses.
- *Sequential Multi-Instance*, this type of activity is similar to the parallel multi-instance, with the only difference that the instances of the activity are executed one after the other (*sequentially*) instead of concurrently. Each activity (of the sequence) is carried on by the same type of resource. Figure 5 shows an example of the student assignments marking, the teacher marks an assignment after the other until all are marked.
- *Simple Loop*, it is used to model an activity that can be executed potentially an unlimited number of times before the process execution continues with the next activity. Unlike the sequential multi-instance activity, the simple loop activity does not take as input a collection of objects. Instead, the simple loop activity is executed until a certain condition becomes true, also called loop exit-condition. Figure 5 shows the example of choosing an available username when registering an account online, the user is asked to enter an username until the chosen one is available.
- *Collapsed Subprocess*, it is used to model a place holder for a process nested within the main process, which is called *subprocess*. A collapsed subprocess is not a single activity but a process itself, with its own start and end events. It is usually represented collapsed either for enhancing the process model simplicity or for lack of information (i.e. it is not known what happens inside the subprocess). Figure 6 shows two views of the same process model: the first (above) having a collapsed subprocess; and the second (below) having the expanded subprocess. Also, a collapsed subprocess may be simple, parallel multi-instance, sequential multi-instance, or simple loop.

Differently from the events, an activity (regardless of its type) is triggered as soon as the process execution flow reaches it, whilst its execution is not immediate.

Sequence Flows. They allow us to define the order in which activities (events) are executed (consumed). The sequence flow shape is an arrow connecting any two BPMN elements (e.g. two activities, an activity and an event, etc.), namely source of the sequence flow and target of the sequence flow. Sequence flows can also be labelled to highlight data conditions in the case of exclusive (or inclusive) choices, as we have seen in Figure 3. Furthermore, a sequence flow can be active or inactive. A sequence flow is active if: (i) its source has been triggered

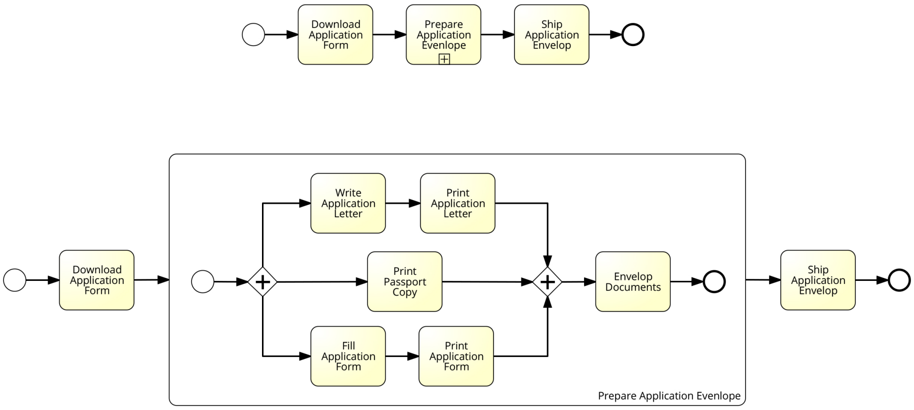


Figure 6: Collapsed and expanded subprocess (resp. above and below).

and executed (consumed) completely; and (ii) its target is not yet been triggered. Otherwise, the sequence flow is inactive.

Gateways. A gateway is a control flow element, and it can be used to model several process behaviors: concurrent executions; tasks synchronization; exclusive choices; and inclusive choices. The gateway shape is a diamond, whose inner symbol identifies the behaviour captured by the gateway. The number of its incoming and outgoing sequence flows determines whether the gateway is a split or a join. Specifically, a gateway is a *split* if the number of its outgoing sequence flows is two or more; or a *join* if the number of its incoming sequence flows is two or more. Figure 7 shows some common gateways in BPMN.

- *XOR Gateways*, they are used to model exclusive choices, their symbol is $\times$. The *XOR-split* represents an exclusive choice, and only one outgoing sequence flow can be activated to continue the process execution flow, the choice is usually based on data conditions (captured as labels of the outgoing sequence flows). The *XOR-join* is used to merge incoming sequence flows, and it lets the process execution flow continue as soon as one incoming sequence flow is active.
- *AND Gateways*, they are used to model concurrent behaviour, their symbol is $+$. The *AND-split* starts a parallel execution of activities, all its outgoing sequence flows are activated at the same time and the process execution flow continues concurrently on all the outgoing sequence flows. The *AND-join* is used to synchronize incoming sequence flows, and it lets the execution continue only when *all* the incoming sequence flows are active.
- *OR Gateways*, they are used to model inclusive choices, their symbol is $\bigcirc$. The *OR-split* represents an inclusive choice, and it allows the concurrent activation of any of its outgoing sequence flows (from just one to all of them), the choice is driven by data, similarly to the XOR-split case. The *OR-join* is used to synchronize incoming sequence flows, and it lets the execution continue only when all the incoming sequence flows that can eventually be

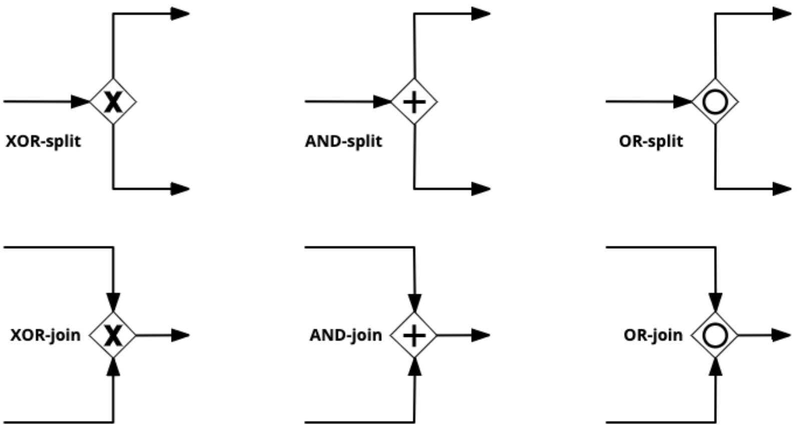
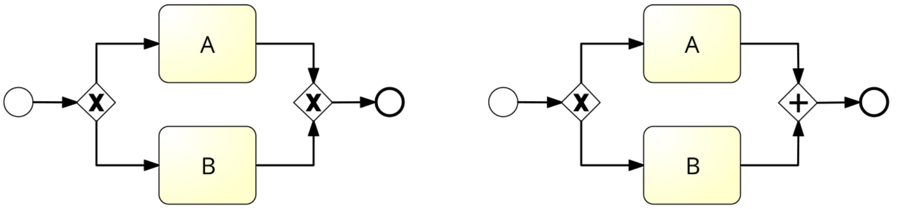


Figure 7: Samples of gateways.



(a) Sound process. (b) Unsound process, *no option to complete*.

Figure 8: Example of proper and improper use of XOR gateways.

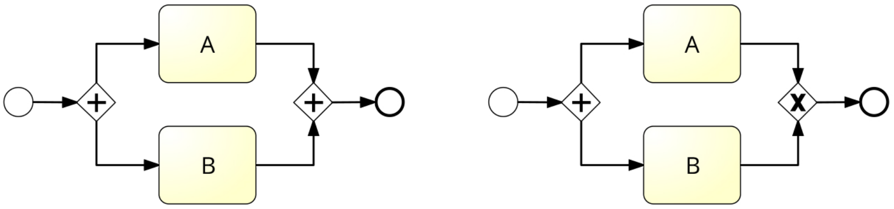
activated are active.

Gateways play an important role for the soundness of the process model. A BPMN process model with one start and one end event is *sound* if and only if the following three properties hold altogether:

- **Option to complete**: any arbitrary process execution can reach the end event;
- **Proper completion**: no end events can be triggered more than once during a process execution;
- **No dead activities**: for each activity of the process, there exists at least one process execution that triggers the activity.

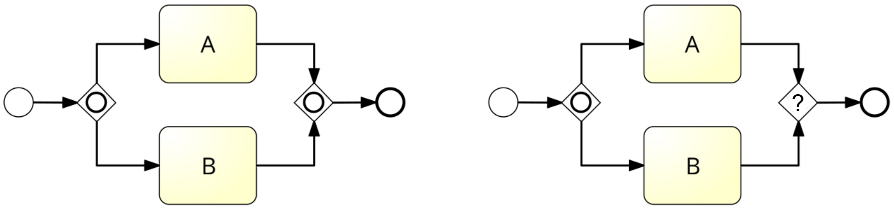
If one of the three properties is violated, the process is unsound and ultimately incorrect. If the gateways are used incorrectly they can generate lack of synchronization and deadlocks, producing unsound models. In the following, we report a few examples of correct and incorrect use of gateways.

Figure 8a shows a sound process model, whose possible executions are: $\langle start, A, end \rangle$; and $\langle start, B, end \rangle$. Instead, Figure 8b shows a process model violating the *option to complete* property, due to a deadlock. The AND-join will wait forever to synchronize the activities A and B, which are mutually exclusive.



(a) Sound process. (b) Unsound process, *no proper completion*.

Figure 9: Example of proper and improper use of AND gateways.



(a) Sound process. (b) Unsound process.

Figure 10: Example of proper and improper use of OR gateways.

Figure 9a shows a proper use of the AND gateways, which allow the concurrent execution of the activities A and B, in this case the possible process executions are:
$\langle start, A, B, end \rangle$; and $\langle start, B, A, end \rangle$. Instead, the process in Figure 9b is unsound because it violates the *proper completion* property. Indeed, when the end event is reached, either activity A or activity B will still be executing because of the lack of synchronization.

The last example takes into account the OR gateways. In Figure 10a, we have a sound process that allows the following executions:
$\langle start, A, B, end \rangle$; $\langle start, B, A, end \rangle$; $\langle start, A, end \rangle$; and $\langle start, B, end \rangle$. In Figure 10b, instead, we left a blank join gateway to show that regardless of its type (either AND or XOR) the process would be unsound. If the join is an AND the process would have a deadlock every time the OR-split activates only one of its outgoing sequence flows. If the join is an XOR the process would have a lack of synchronization every time the OR-split activates both outgoing flows.

The remaining BPMN elements such as *pools*, *swimlanes*, *data objects*, *message flows*, and *annotations*, are used to model respectively: organizations, resources, documents, communications, and notes. Since these elements (and some extensions of the core elements) are not of interest for our research project, we won't provide further details. Instead, we redirect to the BPMN 2.0 specifications [22].

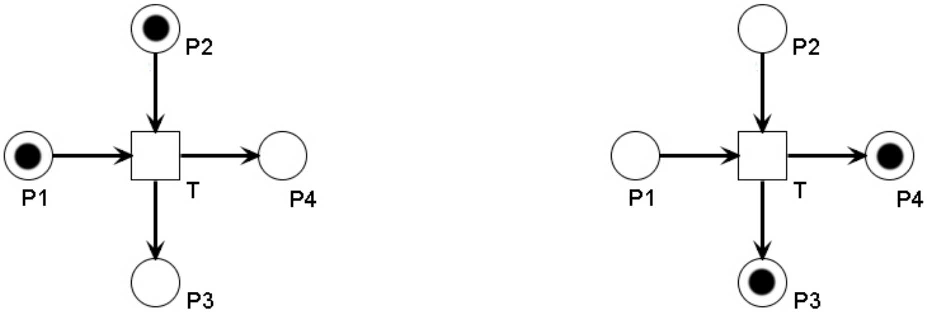
2.2.2. Petri Nets

Petri nets (PNs) take their name from their creator, Carl Adam Petri, who initially designed them to describe chemical processes. Over the last century, Petri nets have been used in different contexts to model anything from algorithms to business processes. Hereinafter, we assume PNs represent business processes, and we describe them accordingly, postponing any formal definition to the chapters that require it.

The PNs graphical elements are only four: places, transitions, arcs, and tokens (Figure 11a shows all of them). A *place* allows us to model a state of the process, its shape is a blank circle. A *transition* represents an activity of the process, its shape is a rectangle. A transition can be either *visible* (if it has a label) or invisible, the latter is usually represented with a black rectangle. *Arcs* connect transitions to places and vice-versa but they cannot connect two places or two transitions. Lastly, *tokens* are used to mark the current state of the process, a token shape is a black large dot. At any time one or more places may contain one or more tokens, the current state of the process is captured by the number of tokens in each place, such information is known as *marking*. A *marking* is a tuple of positive integers having size equal to the number of places of the PN, each integer of the tuple captures the number of tokens in a specific place of the PN. A PN should always have one initial marking and one final marking, which represent the initial state of the process and the final state of the process. As an example, the initial marking of the PN in Figure 11 is: $(1,1,0,0)$; whilst, its final marking is: $(0,0,1,1)$.

Tokens are fundamental for the execution of a Petri net. In order for a transition (activity) to *fire* (be executed), it must be *enabled*. A transition is said to be *enabled* if and only if there is at least one token in each place of the transition's *pre-set*. The *pre-set* of a transition is the set containing the places connected to the transition with an outgoing arc, in Figure 11a the pre-set of the transition T is $\{P1,P2\}$. When a transition is fired, one token is removed from each place of the transition's pre-set, and one token is added to each place of the transition's *post-set*. The *post-set* of a transition is the set containing the places connected to the transition with an incoming arc, in Figure 11a the post-set of the transition T is $\{P3,P4\}$. When a transition is fired, the marking of the Petri net (usually) changes, as well as the state of the process. Figure 11 shows the states (i.e. markings) of a Petri net before and after the transition T is fired.

Even having fewer and simpler elements, PNs have the same semantic power of BPMN models (if not greater), and allow the representation of almost any kind of process behavior. However, the complexity of the processes modelled as PNs is usually higher than BPMN process models, because to represent complex behavior, PNs requires more graphical elements than BPMN. As a supporting example for this latter statement, we report in Figure 12 the PN representation of the Amazon order-to-cash process previously showed in Figure 3, noting that 50% more graphical elements were necessary to model the process with a PN.



(a) Petri net, initial state. (b) Petri net, final state.

Figure 11: Example of Petri net elements, and firing transition.

Furthermore, whilst in the case of the BPMN models the process execution flow is driven by the semantic of the BPMN elements, in the case of PNs, the process execution flow is driven by an overarching execution semantic defined on the PNs' graphical elements.

Finally, the notion of soundness is defined also for processes represented as PNs. A PN with an initial marking *i* and a final marking *o* is sound if and only if the following three properties hold altogether:

- **Option to complete**: starting from the initial marking *i*, any arbitrary process execution can reach the final marking *o*;
- **Proper completion**: none of the allowed process executions can reach markings (i.e. states of the process) having more than one token in a single place;
- **No dead transitions**: for each transition of the PN, there exists a marking (i.e. a state of the process) where the transition is enabled;

As for BPMN process models, any process represented as a PN should be sound.

2.2.3. Other Process Modelling Languages

Even though BPMN and PNs are the most popular modelling languages for business processes, they are not the only available. Especially in the context of automated process discovery, processes are often represented as directed graphs (matrices), trees, or behavioral rules (i.e. *declarative models*).

Directly-Follows Graphs (DFGs) are widely employed in commercial tools[3] Celonis,[4] Minit,[5] and myInvenio.[6] and by many APDAs [8, 12] to capture the process behavior recorded in its execution data (i.e. its event log) in the form of a

---

[3] `http://fluxicon.com/disco`

[4] `http://www.celonis.com`

[5] `http://minitlabs.com/`

[6] `http://www.my-invenio.com`

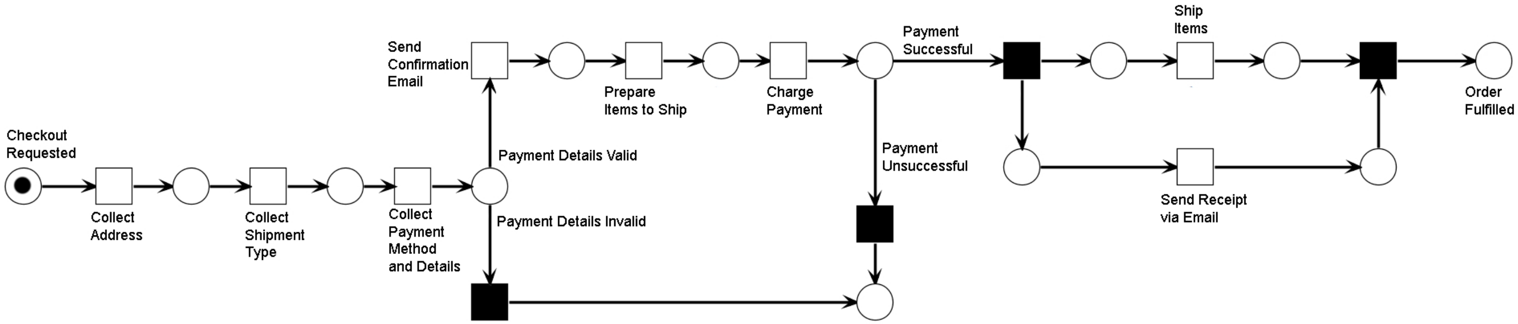


Figure 12: Amazon *order-to-cash* business process (customer perception), as a Petri net.

directed graph. Each node of the DFG represents an event recorded in the event log. Whilst each edge of the DFG connects two events that were observed in at least one process execution as consecutive events, the preceding event being the source of the edge and the succeeding event being the target of the edge. However, despite being very simple and intuitive, this process representation does not provide any execution semantic. Also, as we will see later in this thesis, DFGs are a highly lossy representation of event logs (cf. Chapter 6).

Process Trees capture the processes in a structured manner. Each leaf of the tree represents an activity (event), while each node of the tree represents a group of activities with a specific execution semantic, e.g. sequential or parallel execution, exclusive or inclusive choice. Differently than DFGs, process trees represent clearly and unambiguously the process execution semantic. However, the interpretation of the process trees graphical representation is not intuitive so that it is convenient to convert them into either PNs or BPMN models, such conversion is straightforward and deterministic. Indeed, process trees are equivalent to so-called block-structured BPMN process models with only XOR and AND gateways. Specifically a block-structured BPMN model is one where every split gateway has a corresponding join gateway, such that the fragment of the process model between the split gateway and the join gateway is a Single-Exit Single-Exit (SESE) region [26]. Consequently, process trees are sound by construction.

Declare Models capture the processes' behavior through a set of rules, also known as (Declare) constraints. Each constraint describes a specific detail of the process behavior, for example: *activity X is executed always after activity Y*; or *every time activity X is performed, activity Z is not*. Declare constraints can either be listed in natural language or modelled with ad-hoc graphical elements, usually the former representation is preferred because when the number of constraints is high, a graphical representation would be complex to navigate. Even though each Declare constrain is precise, capturing the whole process behavior in a Declare model can be very difficult, especially because Declare models do not give any information about "undeclared" behavior, e.g. any behavior that does not break the Declare constraint is allowed behavior. Declarative models belong to the *imperative* modelling languages category, as opposed to all the modelling languages we mentioned above, which belong to the *procedural* modelling languages category.

2.3. Process Discovery

Process discovery is an important phase of the BPM lifecycle, since to perform any sort of process analysis it is necessary to start from a process representation, e.g. a process model, that needs to be discovered [1]. However, process discovery is not synonymous of process modelling, but it is a longer operation that evolves over four macro steps [1]. The first step is the *setup*, the company willing to discover one (or more) of its business processes should build a team of domain experts (e.g. employees involved in the process execution), who should collaborate with the BPM analysts. The domain experts aid is fundamental, since the BPM analysts may not be confident with the company environment and business. The second step is the *data collection*, information about the process is gathered, this may include process documentation (either written or verbal) and/or the process execution data recorded in the company IS. The third step is the *process modelling*, the data collected in the previous step is used by the BPM analysts to draw the process model. Lastly, the fourth step is the *quality check*, the process model is presented to the stakeholders to verify its compliance and quality.

Process discovery can be performed applying different methodologies, the most popular are the following three [1].

- *Interview-based discovery*: the BPM team schedules a series of interviews with the domain experts and the employees involved in the process, to understand the process dynamics and its execution. Successively, the information gathered during the interviews is used to draw the process model. This latter is ultimately validated by the interviewees.
- *Workshop-based discovery*: the BPM team, the process owners, the domain experts, and the employees involved in the process, gather together for one (or more) workshops. Each workshop is a team effort for understanding and drawing the process, and checking its quality. This approach ensures a real-time contribution by all the stakeholders, who can share their process' perspectives and give immediate feedback to the BPM team output.
- *Evidence-based discovery* consists in analysing concrete evidences of the process execution in order to discover its model. Evidences usually belong to one of the following categories: (i) written process documentation; (ii) direct observation of the process execution (or role playing within the process execution); (iii) process execution data.
  - *Written process documentation* is most of the times available, however, it could be outdated or it may reflect the ideal process execution, i.e. showing only how the process should execute.
  - *Direct observation* is frequently biased by the fact that employees tend to act differently when observed (i.e. Hawthorne effect), and only the best scenarios may materialise. Playing a role within the process execution (e.g. acting as a customer) is extremely useful to understand the process dynamics and performance, however, it provides only one perspective of the whole process.
  - *Process execution data* is exploited in **automated process discovery**, which is usually quicker and more objective than other process discovery methodologies, since the process model is discovered through a software application that automatically generates the process model from the observed business process execution data.

Out of all these methodologies, *automated process discovery* is the one that attracted the greatest interest in the past decades, as we shall see in the next chapter. However, even though automated process discovery brings many benefits and facilitates the process discovery phase of the BPM lifecycle, such a methodology is not free of drawbacks, which pose big challenges to those who want to overtake them.

3. STATE OF THE ART

Process mining allows analysts to exploit event logs of historical executions of business processes to extract insights regarding the quality, compliance, and performance of these processes. One of the most widely studied process mining operations is automated process discovery. This Chapter [1] is centred on automated process discovery, we discuss what is the goal of this operation, why our research project focuses on it, and we report an extensive literature review, covering the last seven years of research studies addressing the problem of automated process discovery.

3.1. Automated Process Discovery

Automated process discovery is not only a process discovery methodology but also one of the cornerstone operations of tactical process mining. An automated process discovery approach (APDA) is an implementation of such an operation, which takes as input the execution data of a business process, and it outputs a *process model* that captures the control-flow relations between the events that are observed in the input data.

The execution data of a business process is (usually) stored in the form of an *event log*. Formally, an event log is a chronological sequence of events observed during the process executions. In the context of this thesis, the chronological sequence of events produced by the same process instance is referred as trace. An event recorded in the event log represents the happening of something, e.g. an activity is performed, an email is received. An event must have at least three attributes: the identifier of the process case, namely *case ID*; the event name; and the timestamp. However, the data associated to an event can go far beyond these three attributes. For example, an event can also report the resource that generated it, the department of the resource, or the cost associated. There is, indeed, no limit to the type and number of attributes attached to an event, and they can change depending on the context. In general, the more information is collected in an event log, the more insights can be extracted.

To be useful, automatically discovered process models should reflect the behaviour recorded in the event log. Specifically, according to van der Aalst [27], the process model discovered from an event log should be able to: (i) generate each trace of the event log; (ii) generate traces that even not being recorded in the event log are likely to be produced by the process that generated the event log; and (iii) not generate other traces. The first property is called *fitness*, the second *generalization*, and the third *precision*. In addition, the discovered process model should be as simple as possible, a property that is usually quantified via *complexity* measures, and, most important, the model should be sound. *Fitness*,

---

[1] Corresponding to [13].

precision, *generalization*, *complexity*, and *soundness* are measures that allow us to quantify the quality of the discovered process model without the help of the domain experts.

The problem of automated discovery of process models from event logs has been intensively researched in the past two decades. Despite a rich set of proposals, state-of-the-art APDAs suffer from at least one of the following three limitations when receiving in input large and complex event logs: (i) they achieve limited accuracy (i.e. low scores of fitness and/or precision); (ii) they are computationally inefficient to be used in practice; (iii) they discover unsound process models. In the following, we address our **RQ1** (*what are the state-of-the-art APDAs, their strengths and limitations?*), reporting our review of the state-of-the-art APDAs, and showing their features, strengths, and weaknesses by analysing them and (in Chapter 4) by comparing them through the benchmark we designed.

3.2. Methodology

In order to identify and analyse research studies addressing the problem of automated (business) process discovery from event logs, we conducted a *Systematic Literature Review* (SLR) through a scientific, rigorous and replicable approach as specified by Kitchenham [28]. First, we formulated a set of research questions to scope the search, and developed a list of search strings. Next, we ran the search strings on different academic databases. Finally, we applied inclusion criteria to select the studies retrieved through the search.

3.2.1. SLR Research Questions

The objective of our SLR is to identify and analyse research studies addressing the problem of automated (business) process discovery from event logs. Studies related to event log filtering, enhancement, and decomposition are orthogonal to automated process discovery, therefore, we hold them out of our SLR. With this aim, we formulated the following research questions, which can be seen as a break-down of our *RQ1*:

RQ1.1 What approaches exist for *automated* (business) process discovery from *event logs*?
RQ1.2 What *type of process models* can be discovered by these APDAs, and in which *modeling language*?
RQ1.3 Which *semantics* can be captured by a model discovered by these APDAs?
RQ1.4 What *tools* are available to support these APDAs?
RQ1.5 What *type of data* has been used to evaluate these APDAs, and from which application domains?

RQ1.1 is the core research question, which aims at identifying existing APDAs. The other questions allow us to identify a set of classification criteria. Specifically, RQ1.2 categorizes the output of an APDA on the basis of the type of process model discovered (i.e., procedural, declarative or hybrid), and the specific modeling language employed (e.g., Petri nets, BPMN, Declare). RQ1.3 delves into the specific semantic constructs supported by an APDA (e.g., exclusive choice, parallelism, loops). RQ1.4 explores what tools support the different APDAs, while RQ1.5 investigates how the APDAs have been evaluated and in which application domains.

3.2.2. Search String Design

To identify the research studies that would allow us to answer our research questions, we designed four search strings as follows. First, we determined that the term “process discovery” is central in our search. In fact, the research community of process mining and business process management, as well as two of the most cited academic reference books [1, 7], agree that the ability of an algorithm or approach to discover a process model from any source of information regarding the process execution (e.g. an event log) is referred to as *process discovery*. In light of this, we set “process discovery” as first and main search string, as the most general term it would allow us to identify the majority of the APDAs. Then, we selected “learning” and “workflow” as synonyms of “discovery” and “process” (respectively). The term *workflow* because it is used to refer to a procedure executed within a business environment, the term is also known for the corresponding modelling language: *workflow charts*. The term *learning* because it is used in computer science referring to algorithms and techniques that can mock the cognitive function of human beings to derive knowledge, in our context the capacity of discovering/modelling a process model. This led to the following four search strings: (i) “process discovery”, (ii) “workflow discovery”, (iii) “process learning”, (iv) “workflow learning”.

We intentionally excluded the terms “automated”, “automating”, “event log” and “log" from the search strings, because these terms are often not explicitly used. However, this led to retrieving many more studies than those actually focusing on automated process discovery, e.g., studies on process discovery via workshops or interviews. Thus, if a query on a specific data source returned more than one thousand results, we refined it by combining the selected search string with the term “business” or “process mining” to obtain more focused results, e.g., “process discovery AND process mining” or “process discovery AND business”. According to this criterion, the final search strings used for our search were the following:

i. “process discovery AND process mining”
ii. “process learning AND process mining”
iii. “workflow discovery”
iv. “workflow learning”

First, we applied each of the four search strings to Google Scholar, retrieving studies based on the occurrence of one of the search strings in the title, the keywords or the abstract of a paper. Then, we used the following six popular academic databases: Scopus, Web of Science, IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, to double check the studies retrieved from Google Scholar. We noted that this additional search did not return any relevant study that was not already discovered in our primary search. The search was completed in December 2017.

3.2.3. Study Selection

As a last step, as suggested by [29–32], we defined inclusion criteria to ensure an unbiased selection of relevant studies. To be retained, a study must satisfy all the following inclusion criteria.

IN1 *The study proposes an approach for automated (business) process discovery from event logs.* This criterion draws the borders of our search scope and it is direct consequence of RQ1.1.

IN2 *The study proposes an APDA that has been implemented and evaluated.* This criterion let us exclude APDAs whose properties have not been evaluated nor analysed.

IN3 *The study is published in 2011 or later.* Earlier studies have been reviewed and evaluated by De Weerdt et al. [3], therefore, we decided to focus only on the successive studies. Nevertheless, we performed a mapping of the studies assessed in 2011 [3] and their successors (where applicable), cf. Table 2.

IN4 *The study is peer-reviewed.* This criterion guarantees a minimum reasonable quality of the studies included in this SLR.

IN5 *The study is written in English.*

α, α^{+}, α^{++} [33–35]	$\alpha\$$ [36]
AGNEs Miner [37]	—
(DT) Genetic Miner [38,39]	Evolutionary Tree Miner [10]
Heuristics Miner [40,41]	Heuristics Miner [8,42–44]
ILP Miner [45]	Hybrid ILP Miner [46]

Table 2: APDAs assessed by De Weerdt et al. [3] (left) and the respective successors (right).

Inclusion criteria IN3, IN4 and IN5 were automatically applied through the configuration of the search engines. After the application of the latter three inclusion criteria, we obtained a total of 2,820 studies. Then, we skimmed title and abstract of these studies to exclude those studies that were clearly not compliant with IN1. As a result of this first iteration, we obtained 344 studies.

Then, we assessed each of the 344 studies against the inclusion criteria IN1 and IN2. The assessment of IN1 was based on the accurate reading of the abstract, introduction and conclusion. On the other hand, to determine whether a study fulfilled IN2, we relied on the findings reported in the evaluation of the studies. As a result of the iterations, we found 86 studies matching the five inclusion criteria.

However, many of these studies refer to the same APDA, i.e., some studies are either extensions, optimization, preliminaries or generalization of another study. For such reason, we decided to group the studies by either the last version or the most general one: the one capturing the main idea of the APDA, rather than a specific variant, extension, or adaptation. At the end of this process, as shown in Table 3, 34 main groups of discovery algorithms were identified. Figure 13 shows how the studies are distributed over time. We can see that the interest in the topic of automated process discovery grew over time with a sharp increase between 2013 and 2014, and lately reducing to the average number of studies per year.

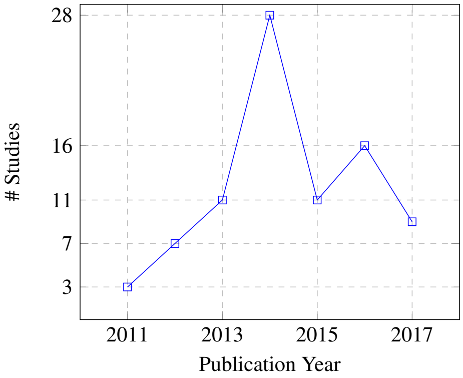


Figure 13: Number of studies over time.

3.3. Classification of the Studies

Driven by the research questions defined in Section 3.2.1, we identified the following classification dimensions to survey the APDAs described in the primary studies:

1. Model type (procedural, declarative, hybrid) and model language (e.g., Petri nets, BPMN, Declare) — RQ1.2
2. Semantic captured in procedural models: parallelism (AND), exclusive choice (XOR), inclusive choice (OR), loop — RQ1.3
3. Type of implementation (standalone or plug-in, and tool accessibility) — RQ1.4
4. Type of evaluation data (real-life, synthetic or artificial log, where a synthetic log is one generated from a real-life model while an artificial log is one generated from an artificial model) and application domain (e.g., insurance, banking, healthcare) — RQ1.5

This information is summarized in Table 3. Each entry in this table refers to the main study of the 34 groups found. Also, we cited all the studies that relate to the main one. Collectively, the information reported by Table 3 allows us to answer RQ1.1, i.e. “what APDAs exist?”

In the remainder, we proceed with surveying each main study along the above classification dimensions, to answer the other research questions.

3.3.1. Model Type and Language (RQ1.2)

The majority of the APDAs (25 out of 34) produce procedural models. Six approaches [48,55,67,70,96,106] discover declarative models in the form of Declare constraints, whereas [77] produces declarative models using the WoMan formalism. The approaches in [83,89] discover hybrid models as a combination of Petri nets and Declare constraints.

Regarding the modeling language of the automatically discovered process models, we notice that Petri nets is the predominant one. However, more recently, we have seen the appearance of APDAs that produce models in BPMN, which is more practice-oriented and less technical than Petri nets. Other technical languages employed, besides Petri nets, include Causal nets, State machines and simple Directed Acyclic Graphs, while Declare is the most commonly-used language when producing declarative models.

Petri nets. In [47], the authors describe an algorithm to extract block-structured Petri nets from event logs. The algorithm works by first building an adjacency matrix between all pairs of tasks and then analyzing the information in it to extract block-structured models consisting of basic sequence, choice, parallel, loop, optional and self-loop structures as building blocks. The approach has been implemented in a standalone tool called HK.

The APDA presented in [36] is based on the $\alpha\$$ algorithm, which can discover invisible tasks involved in non-free-choice constructs. The algorithm is an extension of the well-known α algorithm, one of the very first algorithms for automated process discovery, originally presented in [122].

In [114], the authors propose a generic divide-and-conquer framework for the discovery of process models from large event logs. The approach consists in partitioning the event log into smaller logs and discovering a model from each of them. The output is then assembled from all the models discovered from the sublogs. This APDA aims to produce high quality models by reducing the overall complexity. A range of preliminary studies [115–119] widely illustrate the idea of splitting a large event log into a collection of smaller logs to improve the performance of a discovery algorithm.

van Zelst et al. [46,120,121] propose an improvement of the ILP miner implemented in [45], their approach is based on hybrid variable-based regions. Through hybrid variable-based regions, it is possible to vary the number of variables used within the ILP problems being solved. Using a different number of variables has an impact on the average computation time for solving the ILP problem.

In [98,99], the authors propose an approach that allows the discovery of Petri nets using the theory of grammatical inference. The algorithm has been implemented as a standalone application called RegPFA.

The approach proposed in [108] is based on the observation that activities with no dependencies in an event log can be executed in parallel. In this way, this algorithm can discover process models with concurrency even if the logs fail to meet the completeness criteria. The APDA has been implemented in a tool called ProM-D.

In [76], the authors propose the use of numerical abstract domains for discovering Petri nets from large logs while guaranteeing formal properties of the discovered models. The APDA guarantees the discovery of Petri nets that can reproduce every trace in the log and that are minimal in describing the log behavior.

The approach introduced in [109] addresses the problem of discovering sound Workflow nets from incomplete logs. The method is based on the concept of invariant occurrence between activities, which is used to identify sets of activities (named conjoint occurrence classes) that can be used to infer the behaviors not exhibited in the log.

In [104], the authors leverage data carried by tokens in the execution of a business process to track the state changes in the so-called token logs. This information is used to improve the performance of standard discovery algorithms.

Process trees. The Inductive Miner [60] and the Evolutionary Tree Miner [10] are both based on the extraction of process trees from an event log. Concerning the former, many different variants have been proposed during the last years, but its first appearance was in [61]. Successively, since the original approach was unable to deal with infrequent behavior, an improvement was proposed in [60], which efficiently drops infrequent behavior from logs, still ensuring that the discovered model is behaviorally correct (sound) and highly fitting. Another variant of the Inductive Miner is presented in [9]. This variant can minimize the impact of incompleteness of the input logs. In [63], the authors discuss ways of systematically treating lifecycle information in the discovery task. They introduce a process discovery technique that is able to handle lifecycle data to distinguish between concurrency and interleaving. The approach proposed in [62] provides a graphical support for navigating the discovered model, whilst the one described in [66] can deal with cancelation or error-handling behaviors (i.e., with logs containing traces that do not complete normally). Finally, the variant presented in [64] and [65] combines scalability with quality guarantees. It can be used to mine large event logs and produces sound models.

In [10], Buijs et al. introduce the Evolutionary Tree Miner. This APDA is based on a genetic algorithm that allows the user to drive the discovery process based on preferences with respect to the four quality dimensions of the discovered model: fitness, precision, generalization and complexity. The importance of these four dimensions and how to address their balance in process discovery is widely discussed in the related studies [71–75].

Causal nets. Greco et al. propose an APDA that returns causal nets [90, 91]. A causal net is a net where only the causal relation between activities in a log is captured. This approach encodes causal relations gathered from an event log and if available, background knowledge in terms of precedence constraints over the topology of the resulting model. A discovery algorithm is formulated in terms of reasoning problems over precedence constraints.

In [92], the authors propose an APDA using Maximal Pattern Mining where they discover recurrent sequences of events in the traces of the log. Starting from these patterns they build process models in the form of causal nets.

ProDiGen, a standalone APDA by Vazquez et al. [94, 95], allows users to discover causal nets from event logs using a genetic algorithm. The algorithm is based on: (i) a fitness function that takes into account completeness, precision, and complexity; and (ii) ad-hoc crossover and mutation operators.

Another approach that produces causal nets is the Proximity Miner, presented in [110, 111]. This APDA extracts behavioral relations between the events of the log which are then enhanced using inputs from domain experts.

Finally, in [113], the authors propose an approach to discover causal nets that optimizes the scalability and interpretability of the outputs. The process under analysis is decomposed into a set of stages, such that each stage can be mined separately. The proposed technique discovers a stage decomposition that maximizes modularity.

State machines. The CSM Miner, discussed in [102, 103], discovers state machines from event logs. Instead of focusing on the events or activities that are executed in the context of a particular process, this approach concentrates on the states of the different process perspectives and discover how they are related with each other. These relations are expressed in terms of Composite State Machines (CSM). The CSM Miner provides an interactive visualization of these multi-perspective state-based models.

BPMN models. In [101], Conforti et al. present the BPMN Miner, an APDA for discovering BPMN models containing sub-processes, activity markers such as multi-instance and loops, and interrupting and non-interrupting boundary events (to model exception handling). The approach has been subsequently improved in [100] to make it robust to noise in event logs.

Another APDA for discovering BPMN models has been presented in [93]. In this approach, a hierarchical view on process models is formally specified and an evolution strategy is applied on it. The evolution strategy, which is guided by the diversity of the process model population, efficiently finds the process models that best represent a given event log.

A further approach to discover BPMN models is the Dynamic Constructs Competition Miner [84, 86, 87]. This APDA extends the Constructs Competition Miner presented in [85], and it is based on a divide-and-conquer algorithm which discovers block-structured process models from logs.

In [8], the authors present the Flexible Heuristics Miner. This approach can discover process models containing non-trivial constructs, but with a low degree of block structuredness. The discovered models are a specific type of Heuristics nets where the semantics of splits and joins is represented using split/join frequency tables. This results in easy to understand process models even in the presence of non-trivial constructs and log noise. The discovery algorithm is based on the original Heuristics Miner [41]. Flexible Heuristics Miner was later improved, as anomalies were found concerning the validity and completeness of the resulting process model. The implementation of the improvements led to the Updated Heuristics Miner [44]. Successively, a data-aware version of the Heuristics Miner that takes into consideration data attached to events in a log was presented in [43]. Whilst, in [11, 42], the authors propose an improvement for Heuristics Miner based on the idea of separating the objective of producing accurate models and that of ensuring their structuredness and soundness. Instead of directly discovering a structured process model, the approach first discovers accurate, possibly unstructured (and unsound) process models, and then transforms the resulting process model into a structured (and sound) one.

Fodina [12, 112] is an APDA based on the Heuristics Miner [41]. However, differently from the latter, Fodina is more robust to noisy data, it is able to discover duplicate activities, and allows for flexible configuration options to drive the discovery according to end user inputs.

Declarative models. In [49], the authors present the first basic approach for mining declarative process models expressed using Declare constraints [123, 124]. This approach was improved in [48] using a two-phase approach. The first phase is based on an apriori algorithm used to identify frequent sets of correlated activities. A list of candidate constraints is built on the basis of the correlated activity sets. In the second phase, the constraints are checked by replaying the log on specific automata, each accepting only those traces that are compliant to one constraint. Those constraints satisfied by a percentage of traces higher than a user-defined threshold, are discovered. Other variants of the same approach are presented in [50–54]. The technique presented in [50] leverages apriori knowledge to guide the discovery task. In [51], the approach is adapted to be used in cross-organizational environments in which different organizations execute the same process in different variants. In [52], the author extends the approach to discover metric temporal constraints, i.e., constraints taking into account the time distance between events. Finally, in [53, 54], the authors propose mechanisms to reduce the execution times of the original approach presented in [48].

MINERful [55–57] discovers Declare constraints using a two-phase approach. The first phase computes statistical data describing the occurrences of activities and their interplay in the log. The second one checks the validity of Declare constraints by querying such a statistic data structure (knowledge base). In [58, 59], the approach is extended to discover target-branched Declare constraints, i.e. constraints in which the target parameter is the disjunction of two or more activities.

The approach presented in [67] is the first for the discovery of Declare constraints with an extended semantics that take into consideration data conditions. The data-aware semantics of Declare presented in this paper is based on first-order temporal logic. The approach presented in [96, 97] is based on the use of discriminative rule mining to determine how the characteristics of the activity lifecycles in a business process influence the validity of a Declare constraint in that process.

Other approaches for the discovery of Declare constraints have been presented in [70, 106]. In [70], the authors present the Evolutionary Declare Miner that implements the discovery task using a genetic algorithm. The SQLMiner, presented in [106], is based on a discovery approach that directly works on relational event data by querying a log with standard SQL. By leveraging database performance technology, the discovery procedure is extremely fast. Queries can be customized and cover process perspectives beyond control flow [107].

The WoMan framework, proposed by Ferilli in [77] and further extended in the related studies [78–82], is based on an algorithm that learns and refines process models from event logs by discovering first-order logic constraints. It guarantees incrementality in learning and adapting the models, the ability to express triggers and conditions on the process tasks and efficiency.

Further approaches. In [68, 69], the authors introduce a monitoring framework for automated process discovery. A monitoring context is used to extract traces from relational event data and attach different types of metrics to them. Based on these metrics, traces with certain characteristics can be selected and used for the discovery of process models expressed as directly-follows graphs.

Vasilecas et al. [88] present an approach for the extraction of directed acyclic graphs from event logs. Starting from these graphs, they generate Bayesian belief networks, one of the most common probabilistic models, and use these networks to efficiently analyze business processes.

In [105], the authors show how conditional partial order graphs, a compact representation of families of partial orders, can be used for addressing the problem of compact and easy-to-comprehend representation of event logs with data. They present algorithms for extracting both the control flow as well as relevant data parameters from a given event log and show how conditional partial order graphs can be used to visualize the obtained results. The approach has been implemented as a Workcraft plug-in and as a standalone application called PGminer.

The Hybrid Miner, presented in [83], puts forward the idea of discovering a hybrid model from an event log based on the semantics defined in [125]. According to such semantics, a hybrid process model is a hierarchical model, where each node represents a sub-process, which may be specified in a declarative or procedural way. Petri nets are used for representing procedural sub-processes and Declare for representing declarative sub-processes.

[89] proposes an approach for the discovery of hybrid models based on the

semantics proposed in [126]. Differently from the semantics introduced in [125], where a hybrid process model is hierarchical, the semantics defined in [126] is devoted to obtain a fully mixed language, where procedural and declarative constructs can be connected with each other.

3.3.2. Procedural Language Constructs (RQ1.3)

All the 25 APDAs that discover a procedural model can detect the basic control-flow structure of sequence. Out of these, only four can also discover inclusive choices, but none in the context of non-block-structured models. In fact, [10, 60] are able to directly identify block-structured inclusive choices (using process trees), while [100, 114] can detect this construct only when used on top of [10] or [60] (i.e. indirectly).

The remaining 21 APDAs can discover constructs for parallelism, exclusive choice and loops, with the exception of: [68], which can detect exclusive choice and loops but not parallelism; [105], which can detect parallelism and exclusive choice but not loops; and [88], which can discover exclusive choices only.

3.3.3. Implementation (RQ1.4)

Over 50% of the APDAs (18 out of 34) provide an implementation as a plug-in for the ProM platform. [2] The reason behind the popularity of ProM can be explained by its open-source, platform-independent, and adaptable framework, which allows researchers to easily develop and test new discovery algorithms. Furthermore, ProM was the first software tool for process mining resulting in the older and most known framework. One of the approaches which has a ProM implementation [55] is also available as standalone tool. The works [42, 100, 113] provide both a standalone implementation and a further implementation as a plug-in for Apromore,[3] which is an online process analytics platform also available under an open-source license. Finally, one APDA [105] has been implemented as a plug-in for Workcraft,[4] a platform for designing concurrent systems.

We note that 21 tools out of 34 are made publicly available to the community, leaving out 4 ProM plug-ins and 9 standalone tools.

3.3.4. Evaluation Data and Domains (RQ1.5)

The surveyed APDAs have all been evaluated using one or more of the following three types of event logs: (i) real-life logs, i.e. real-life processes execution data; (ii) synthetic logs, generated by replaying real-life process models; and (iii) artificial logs, generated by replaying artificial models.

We found that the majority of the APDAs (30 out of 34) were tested using real-life logs. Among them, 11 approaches (cf. [12, 42, 47, 48, 55, 88–90, 96, 98, 108])

---

[2] `http://promtools.org`

[3] `http://apromore.org`

[4] `http://workcraft.org`

were further tested against synthetic logs. Whilst 13 approaches (cf. [10,36,42, 76,77,84,89,92,93,100,104,105,114]) against artificial logs. Finally, one APDA was tested both on synthetic and artificial logs only (cf. [94]), while [46,109] were tested on artificial logs and [70] on synthetic logs only. Among the approaches tested on real-life logs, we observed a growing trend in using publicly-available logs, as opposed to private logs which hamper the replicability of the results.

Concerning the application domains of the real-life logs, we note that several methods used a selection of the event logs made available by the Business Process Intelligence Challenge (BPIC), which is held annually as part of the BPM Conference series. These logs are publicly available at the *4TU Centre for Research Data*,[5] and cover domains such as healthcare (used by [42,55,60,67,92,106]), banking (used by [42,55,60,83,88,93,98,102,106,114]), IT support management in automotive (cf. [42,96,98]), and public administration (cf. [10,48,60,76,114]). A public log recording a process for managing road traffic fines (also available at the *4TU Centre for Research Data*) was used in [42]. In [105], the authors use logs from various domains available at `http://www.processmining.be/actitrac/`.

Besides these publicly-available logs, a range of private logs were also used, originating from different domains such as logistics (cf. [108,110]), traffic congestion dynamics [90], employers habits (cf. [77,102]), automotive [36], healthcare [42,93], and project management and insurance (cf. [68,100]).

3.4. Threats to Validity

The greatest threat to validity refers to the potential selection bias inaccuracies in data extraction and analysis typical of literature reviews. We tried to minimize such issues by adhering to the guidelines outlined by Kitchenham [28], nonetheless, the inclusion criteria design and the consequently selection of the studies remains prone to subjectiveness. The latter should be considered as a limitation of our work, even though, to the best of our knowledge (and the reviewers of our work knowledge), we believe no fundamental study was left aside. Yet, while we cannot guarantee it, we showed that we used well-known literature sources and libraries in information technology to extract relevant studies on the topic of automated process discovery. We performed a backward reference search to avoid the exclusion of potentially relevant papers, and, to avoid that our review was threatened by insufficient reliability, we ensured that the search process could be replicated by other researchers. However, the search may produce different results as the algorithm used by source libraries to rank results based on relevance may be updated (see, e.g. Google Scholar).

---

[5] `https://data.4tu.nl/repository/collection:event_logs_real`

3.5. Summary

In this chapter, we formally introduced the problem of automated process discovery, and we reported an extensive systematic literature review of the state-of-the-art APDAs.

Our review highlights a growing interest in the field of automated process discovery, and confirms the existence of a wide and heterogeneous number of proposals. Between 2012 and 2017 more than 80 studies proposing an automated process discovery approach have been published. We grouped these studies into 34 groups, each referring to a specific APDA and we described their features, such as: the type of model they can discover, semantic they are able to represent, type of implementation and available artifacts and tools, and type of evaluation. Despite the variety of proposals, we could clearly identify two main streams: approaches that output procedural process models, and approaches that output declarative process models. Furthermore, while the latter ones only rely on declarative statements to represent a process, the former provide various language alternatives, although most of these methods output Petri nets. The predominance of Petri nets is driven by the semantic power of this language, and by the requirements of the methods used to assess the quality of the discovered process models (chiefly, fitness and precision).

The information gathered in our SLR allowed us to answer the first part of our RQ1 (*what are the state-of-the-art APDAs?*), in the next chapter, starting from the outcome of our SLR, we will address the second part of our RQ1, focusing on identifying the strengths and the weakness of the state-of-the-art APDAs.

Method	Main study	Year	Related studies	Model type	Model language	Semantic Constructs				Implementation		Evaluation		
						AND	XOR	OR	Loop	Framework	Accessible	Real-life	Synth.	Art.
HK	Huang and Kumar [47]	2012		Procedural	Petri nets	✓	✓		✓	Standalone		✓	✓	
Declare Miner	Maggi et al. [48]	2012	[49–54]	Declarative	Declare					ProM	✓	✓	✓	
MINERful	Di Ciccio, Mecella [55]	2013	[56–59]	Declarative	Declare					ProM, Standalone	✓	✓	✓	
Inductive Miner - Infrequent	Leemans et al. [60]	2013	[9, 61–66]	Procedural	Process trees	✓	✓	✓	✓	ProM	✓	✓		
Data-aware Declare Miner	Maggi et al. [67]	2013		Declarative	Declare					ProM	✓	✓		
Process Skeletonization	Abe, Kudo [68]	2014	[69]	Procedural	Directly-follows graphs		✓		✓	Standalone		✓		
Evolutionary Declare Miner	vanden Broucke et al. [70]	2014		Declarative	Declare					Standalone			✓	
Evolutionary Tree Miner	Buijs e al. [10]	2014	[71–75]	Procedural	Process trees	✓	✓	✓	✓	ProM	✓	✓		✓
Aim	Carmona, Cortadella [76]	2014		Procedural	Petri nets	✓	✓		✓	Standalone		✓		✓
WoMan	Ferilli [77]	2014	[78–82]	Declarative	WoMan					Standalone		✓		✓
Hybrid Miner	Maggi et al. [83]	2014		Hybrid	Declare + Petri nets					ProM	✓	✓		
Competition Miner	Redlich et al. [84]	2014	[85–87]	Procedural	BPMN	✓	✓		✓	Standalone		✓		✓
Direted Acyclic Graphs	Vasilecas et al. [88]	2014		Procedural	Directed acyclic graphs		✓			Standalone		✓	✓	
Fusion Miner	De Smedt et al. [89]	2015		Hybrid	Declare + Petri nets					ProM	✓	✓	✓	✓
CNMining	Greco et al. [90]	2015	[91]	Procedural	Causal nets	✓	✓		✓	ProM	✓	✓	✓	
alpha$	Guo et al. [36]	2015		Procedural	Petri nets	✓	✓		✓	ProM	✓	✓		✓
Maximal Pattern Mining	Liesaputra et al. [92]	2015		Procedural	Causal nets	✓	✓		✓	ProM		✓		✓
DGEM	Molka et al. [93]	2015		Procedural	BPMN	✓	✓		✓	Standalone		✓		✓
ProDiGen	Vazquez et al. [94]	2015	[95]	Procedural	Causal nets	✓	✓		✓	ProM			✓	✓
Non-Atomic Declare Miner	Bernardi et al. [96]	2016	[97]	Declarative	Declare					ProM	✓	✓	✓	
RegPFA	Breuker et al. [98]	2016	[99]	Procedural	Petri nets	✓	✓		✓	Standalone	✓	✓	✓	
BPMN Miner	Conforti et al. [100]	2016	[101]	Procedural	BPMN	✓	✓	✓	✓	Apromore, Standalone	✓	✓		✓
CSMMiner	van Eck et al. [102]	2016	[103]	Procedural	State machines	✓	✓		✓	ProM	✓	✓		
TAU miner	Li et al. [104]	2016		Procedural	Petri nets	✓	✓		✓	ProM		✓		✓
PGminer	Mokhov et al. [105]	2016		Procedural	Partial order graphs	✓	✓			Standalone, Workcraft	✓	✓		✓
SQLMiner	Schönig et al. [106]	2016	[107]	Declarative	Declare					Standalone	✓	✓		
ProM-D	Song et al. [108]	2016		Procedural	Petri nets	✓	✓		✓	Standalone		✓	✓	
CoMiner	Tapia-Flores et al. [109]	2016		Procedural	Petri nets	✓	✓		✓	ProM				✓
Proximity Miner	Yahya et al. [110]	2016	[111]	Procedural	Causal nets	✓	✓		✓	ProM	✓	✓		
Heuristics Miner	Augusto et al. [42]	2017	[8, 11, 43, 44]	Procedural	BPMN	✓	✓		✓	Apromore, Standalone	✓	✓	✓	✓
Fodina	vanden Broucke et al. [12]	2017	[112]	Procedural	BPMN	✓	✓		✓	ProM	✓	✓	✓	
Stage miner	Nguyen et al. [113]	2017		Procedural	Causal nets	✓	✓		✓	Apromore, Standalone	✓	✓		
Decomposed Process Miner	Verbeek, van der Aalst [114]	2017	[115–119]	Procedural	Petri nets	✓	✓	✓	✓	ProM	✓	✓		✓
HybridILPMiner	van Zelst et al. [46]	2017	[120, 121]	Procedural	Petri nets	✓	✓		✓	ProM	✓			✓

Table 3: Overview of the 34 primary studies resulting from the search (ordered by year and author).

4. BENCHMARK

The majority of the APDAs we surveyed in our SLR have been assessed in an ad-hoc manner, with different authors employing different evaluation datasets, experimental setups, evaluation measures, and baselines. Such a variety in the evaluations of the APDAs leads to incomparable conclusions and sometimes unreproducible results, due to the use of non-publicly available datasets.

In this chapter,[1] we try to fill this gap by providing a benchmark framework designed to enable researchers to empirically compare new APDAs against existing ones in a unified setting. The benchmark is provided as an open-source command-line Java application allowing researchers to replicate the reported experiments with minimal configuration effort. Starting from the outcome of our SLR, the classified inventory of APDAs, we employed our benchmark framework to assess six implementations of representative APDAs, covering twelve publicly-available real-life event logs, twelve proprietary real-life event logs, and nine quality measures covering four quality dimensions for automatically discovered process models: fitness, precision, generalization and complexity; as well as soundness and execution time. The results of our evaluation will allow us to identify the existing APDAs *strengths* and *limitations*.

The rest of the chapter is structured as follows. Section 4.1 presents the APDAs we selected for our benchmark out of the APDAs identified in our SLR. Section 4.2 formally introduces the quality measures we use to assess the APDAs in our benchmark. Section 4.3 describes the event logs included in our benchmark, while Section 4.4 reports the results of our benchmark. Finally, Section 4.7 summarises the chapter.

4.1. APDAs Selection

Assessing all the APDAs that resulted from the search would not be feasible in a single study due to the heterogeneous nature of the inputs required and the outputs produced, which would force us to set up different type of evaluations (e.g. for declarative process models and procedural process models) that would be time demanding and not cross-comparable. Hence, we decided to focus on the largest subset of comparable APDAs. The APDAs considered were the ones satisfying the following criteria:

i an implementation of the APDA is publicly accessible;

ii the output of the APDA is a BPMN model or a Petri net.

APDAs that produce BPMN models were retained because it is a de facto and de jure standard for process modelling, while APDAs that produce Petri nets were also included due to the fact that a large number of existing APDAs produce Petri nets as discussed in Chapter 2.

---

[1] Corresponding to [13].

The application of these criteria resulted in an initial selection of the following APDAs (corresponding to one third of the total studies): $\alpha\$$ [36], Inductive Miner [61], Evolutionary Tree Miner [10], Fodina [12], Structured Heuristic Miner 6.0 [42], Hybrid ILP Miner [121], RegPFA [98], Stage Miner [113], BPMN Miner [100], Decomposed Process Mining [118].

A posteriori, we excluded the latter four due to the following reasons: Decomposed Process Mining, BPMN Miner, and Stage Miner were excluded as such APDAs follow a divide-and-conquer approach which could be applied on top of any other APDA to improve its results; on the other hand, we excluded RegPFA because its output is a graphical representation of a Petri net (DOT), which could not be seamlessly serialized into the standard PNML format. [2]

We also considered including commercial process mining tools in the benchmark. Specifically, we investigated Disco,[3] Celonis,[4] Minit,[5] and myInvenio.[6] Disco and Minit are not able to produce business process models from event logs. Instead, they can only produce directly-follows graphs (DFGs), which do not have an execution semantics. Indeed, when a given node of the DFG has several incoming arcs, the DFG does not tell us whether or not the task in question should wait for all its incoming edges to be active, or just for one of them, or a subset of them. A similar remark applies to split points in the DFG. Given their lack of execution semantics, it is not possible to directly translate a DFG into a BPMN model or a Petri net. Instead, one has to determine what is the intended behavior at each split and join point, which is precisely what several of the APDAs based on DFGs do (e.g., the Inductive Miner or Fodina).

Celonis and myInvenio can produce BPMN process models but all they do is to insert OR (inclusive) gateways at the split and join points of the process map. To the best of our knowledge, there is no existing technique for measuring precision and fitness for BPMN models with OR-joins. When the model does not contain OR-joins, or when the OR-joins are arranged in block-structured topologies, it is possible to translate the BPMN models to Petri nets using existing mappings from BPMN to Petri nets [127]. Once the model is translated as a Petri net, it becomes possible to use existing techniques for assessing fitness and precision available for Petri nets. But when OR-joins appear in arbitrary topologies, including unstructured cycles, this approach cannot be applied.

In conclusion, the final selection of methods for the benchmark contained: $\alpha\$$, Inductive Miner (IM), Evolutionary Tree Miner (ETM), Fodina (FO), Structured Heuristic Miner 6.0 (S-HM$_6$), and Hybrid ILP Miner (HILP).

---

[2] The XML Petri net format.

[3] `http://fluxicon.com/disco`

[4] `http://www.celonis.com`

[5] `http://minitlabs.com/`

[6] `http://www.my-invenio.com`

4.2. Evaluation Measures

For all the selected APDAs, we measured the following accuracy and complexity measures: fitness, precision, generalization, complexity, and soundness.

Fitness measures the ability of a model to reproduce the behavior contained in a log. Under trace semantics, a fitness of 1 means that the model can reproduce every trace in the log. In our benchmark, we use the fitness measure proposed by Adriansyah et al. [128], which measures the degree to which every trace in the log can be aligned (with a small number of errors) with a trace produced by the model. In other words, this measures tells us how close on average a given trace in the log can be aligned with a trace that can be generated by the model.

Precision measures the ability of a model to generate only the behavior found in the log. [7] A score of 1 indicates that any trace produced by the model is contained in the log. In our benchmark, we use the precision measure defined by Adriansyah et al. [129], which is based on similar principles as the above fitness measure. Recall and precision can be combined into a single measure known as F-score, which is the harmonic mean of the two measurements $\left(2 \cdot \frac{Fitness \cdot Precision}{Fitness + Precision}\right)$.

Generalization refers to the ability of an automated discovery algorithm to discover process models that generate traces that are not present in the log but that can be produced by the business process under observation. In other words, an APDA has a high generalization on a given event log if it is able to discover a process model from the event log, which generates traces that: (i) are not in the event log, but (ii) can be produced by the business process that produced the event log. Note that, in the context of this thesis, unlike fitness and precision, generalization is a property of an APDA on an event log, and not a property of the model produced by an APDA when applied to a given event log. In line with the above definition, we use k-fold cross-validation [130] to measure generalization. This k-fold cross-validation approach to measure the ability of an APDA to generalize the behavior observed in the event log, has been advocated in several studies in the field of automated process discovery [27, 131–133]. Concretely, we divide the log into k parts, we discover a process model from $k-1$ parts (i.e., we hold-out one part), and measure the fitness of the discovered model against the part held out. This is repeated for every possible part held out. Generalization is the mean of the fitness values obtained for each part held out. A generalization of one means that the discovered model produces traces in the observed process, even if those traces are not in the log from which the model was discovered. We remark that this definition of generalisation is slightly different than the one given in Section 3.1, since it assesses a quality of the APDA rather than a quality of the process model.

The algorithm we implemented to measure the generalization computes the folds of each log randomly, i.e. starting from the complete log a random set of

---

[7] We note that this definition of precision slightly differ from the one given in Section 3.1, due to its operationalization.

traces is selected to compose each fold. To compare the generalization of different APDAs, we ensured that the folds given as input to each APDA were always the same. In the results reported below, we use a value of $k = 3$ (as opposed to the traditional value of $k = 10$) for the following two reasons. First, for a matter of performance, since the fitness calculation for most of the model-log pairs is slow, and repeating it 10 times for every APDA-log combination is costly. Second, for a matter of effectiveness, since the higher the number of folds the lower is the chance that the test-fold (1 out of K) contains traces not observed in the training folds (K-1 out of K).

Complexity refers to how difficult it is to understand a model. Several complexity metrics have been empirically shown to be (inversely) related to the understandability of process models [14]. These and other empirical findings on process model understandability are distilled in the seven Process Modeling Guidelines (7PMG) compiled by Mendling et al. [134] :

PMG1: *Use as few model elements as possible*; this guideline relates to the *Size* of the process model, which measures the number of nodes.

PMG2: *Minimize the routing paths per element*; this guideline relates to the *Control-Flow Complexity (CFC)* metric [135], which measures the amount of branching induced by the split gateways in a process model.

PMG3: *Use one start event for each trigger and one end event for each outcome.*

PMG4: *Model as structured as possible*; this guideline tells us that for every split gateway in a process model, there should be a corresponding join gateway, such that the sub-graph between the split and the join gateway is a single-entry, single-exit region. This guideline relates to the *structuredness* metric, i.e. the percentage of nodes directly located inside a single-entry single-exit fragment. The more nodes in a process model are located outside such fragments, the lower is the value of the structuredness measure.

PMG5: *Avoid OR gateways where possible.*

PMG6: *Use verb-object activity labels.*

PMG7: *Decompose a model with more than 30 elements.*

In our benchmark, we include the complexity measures that directly relate to guidelines PMG1, PMG2, and PMG4. Guidelines PMG3 and PMG5 are discussed only when they are violated, while guidelines PMG6 and PMG7 are not applicable to our context. Indeed, PMG6 refers to the labelling style of activities: in our case, we take the activity labels directly from the event log. PMG7 only applies to approaches that discover hierarchical process models, while in our case we discover flat models.

In the following we provide the individual formulas used to compute the three complexity measures:

- Size: $S_N(G) = |N|$, where N is the set of nodes of the process model G;
- CFC: $CFC(G) = \sum_{c \in S_{and}} 1 + \sum_{c \in S_{xor}} |c_{xor}\bullet| + \sum_{c \in S_{or}} 2^{|c_{or}\bullet|} - 1$, where S_{and}, S_{xor},

and S_{or} are (respectively) the sets of AND, XOR, and OR gateways of the process model G, and $\bullet$ identifies the activities that are targets of the outgoing sequence flows of the gateways;

- Structuredness: $\Phi = 1 - \frac{S_N(G')}{S_N(G)}$, where G is the original process model and G' is the reduced process model.

Lastly, *soundness* assesses the behavioral correctness of a process model by reporting whether the model violates one of the three soundness criteria [136]: (i) option to complete; (ii) proper completion; and (iii) no dead transitions.

4.3. Setup and Datasets

To guarantee the reproducibility of our benchmark and to provide the community with a tool for comparing new APDAs with the ones evaluated in this thesis, we developed a command-line Java application that performs measurements of accuracy and complexity measures on the APDAs selected above against all the event logs used in our benchmark. The only exception was ETM, which we could not embed in our tool due to its complex configuration settings, hence we relied on its ProM implementation.

Our benchmark tool can be easily extended to incorporate new event logs. Moreover, one can include additional APDAs and quality measures by implementing two predefined interfaces. This is possible through the use of *Java reflection*, which allows the tool to automatically detect the presence of new APDAs and measures. More information about the interfaces to implement, how to include them in the benchmark, and the benchmark source code structure are available in the *readme* files provided with the tool.[8]

For our evaluation, we used two datasets. The first is the collection of real-life event logs publicly available at the 4TU Centre for Research Data as of March 2017.[9] Out of this collection, we considered the *BPI Challenge* (BPIC) logs, the *Road Traffic Fines Management Process* (RTFMP) log, and the *SEPSIS Cases* log. These logs record executions of business processes from a variety of domains, e.g., healthcare, finance, government and IT service management. For our evaluation we held out those logs that do not explicitly capture business processes (i.e., the BPIC 2011 and 2016 logs), and those contained in other logs (e.g., the *Environmental permit application process* log). Finally, in seven logs (i.e., the BPIC14, BPIC15 collection, and BPIC17 logs), we applied the filtering technique proposed in [137] to remove infrequent behavior.[10] This filtering step was nec-

---

[8] The tool and its source code are available at `https://doi.org/10.5281/zenodo.1219321`.

[9] `https://data.4tu.nl/repository/collection:event_logs_real`

[10] This technique uses a parameter called "percentile" which refers to the percentile of the distribution of the frequency of the arcs in the directly-follows graph extracted from the log, to automatically determine the frequency threshold for the filtering. We set this parameter to its default value of 12.5%.

Log Name	Total traces	Dist. traces (%)	Total events	Dist. events	Tr. length		
					min	avg	max
BPIC12	13,087	33.4	262,200	36	3	20	175
BPIC13$_{\text{cp}}$	1,487	12.3	6,660	7	1	4	35
BPIC13$_{\text{inc}}$	7,554	20.0	65,533	13	1	9	123
BPIC14$_{\text{f}}$	41,353	36.1	369,485	9	3	9	167
BPIC15$_{\text{1f}}$	902	32.7	21,656	70	5	24	50
BPIC15$_{\text{2f}}$	681	61.7	24,678	82	4	36	63
BPIC15$_{\text{3f}}$	1,369	60.3	43,786	62	4	32	54
BPIC15$_{\text{4f}}$	860	52.4	29,403	65	5	34	54
BPIC15$_{\text{5f}}$	975	45.7	30,030	74	4	31	61
BPIC17$_{\text{f}}$	21,861	40.1	714,198	41	11	33	113
RTFMP	150,370	0.2	561,470	11	2	4	20
SEPSIS	1,050	80.6	15,214	16	3	14	185

Table 4: Descriptive statistics of public logs.

essary since all the models discovered by the considered APDAs exhibited very poor accuracy (F-score close to 0 or not computable) on the above logs, making the comparison useless.

Table 4 reports the characteristics of the twelve logs used. These logs are widely heterogeneous ranging from small to very large, with a log size ranging from 681 traces (for the BPIC15$_{\text{2f}}$ log) to 150,370 traces (for the RTFMP log). A Similar variety can be observed in the percentage of distinct traces, ranging from 0.2% to 80.6%, and the number of event classes (i.e., activities executed within the process), ranging from 7 to 82. Finally, the length of a trace also varies from very short, with traces containing only one event, to very long with traces containing 185 events.

The second dataset is composed of twelve proprietary logs sourced from several companies around the world. Table 5 reports the characteristics of these logs. Also in this case, the logs are quite heterogeneous, with the number of traces (and the percentage of distinct traces) ranging from 225 (of which 99.9% distinct) to 787,657 (of which 0.01% distinct). The number of recorded events varies between 4,434 and 2,099,835, whilst the number of event classes ranges from 8 to 310.

We performed two types of evaluations. In the first evaluation, we compared all the APDAs using their default parameters. In the second one, we analysed to what extent each APDA could improve its output using grid-search-based (i.e. brute-force) hyper-parameter optimization. We decided to apply a grid-search-based hyper-parameter optimization in order to exhaustively explore the solution space of an APDA. We note that there are no ad-hoc hyper-parameter optimization approaches for all the APDA included in this evaluation. However, due to the extremely-long execution times, it was prohibitive to hyper-parameter optimize $\alpha\$$ and ETM, so we held them out from the second evaluation. Additionally, we

Log Name	Total traces	Dist. traces (%)	Total events	Dist. events	Tr. length min	avg	max
PRT1	12,720	8.1	75,353	9	2	5	64
PRT2	1,182	97.5	46,282	9	12	39	276
PRT3	1,600	19.9	13,720	15	6	8	9
PRT4	20,000	29.7	166,282	11	6	8	36
PRT5	739	0.01	4,434	6	6	6	6
PRT6	744	22.4	6,011	9	7	8	21
PRT7	2,000	6.4	16,353	13	8	8	11
PRT8	225	99.9	9,086	55	2	40	350
PRT9	787,657	0.01	1,808,706	8	1	2	58
PRT10	43,514	0.01	78,864	19	1	1	15
PRT11	174,842	3.0	2,099,835	310	2	12	804
PRT12	37,345	7.5	163,224	20	1	4	27

Table 5: Descriptive statistics of proprietary logs.

excluded HILP since we did not find any input parameters which could be used to optimize the F-score of the models produced. For the remaining three APDAs, we explored the following input parameters: the two filtering thresholds required as input by S-HM$_6$, the only filtering threshold required as input by IM, and the filtering threshold and the boolean flag required as input by FO. All the thresholds ranged from 0.0 to 1.0, though to appreciate variance in the discovered process models, we used steps of 0.05 for IM, steps of 0.10 for the thresholds of FO, and steps of 0.20 for S-HM$_6$. For FO, we considered all the possible combinations of the filtering threshold and the boolean flag. In terms of event logs, in the second evaluation we considered all logs except PRT11, because all the APDAs failed to generate a model from this log (except ETM), as evidenced by the results of the first evaluation with default parameters.

We performed the first evaluation on a 6-core Intel Xeon CPU E5-1650 v3 @ 3.50GHz with 128GB RAM running Java 8. We allocated a total of 16GB to the heap space and 10GB to the stack space. We enforced a timeout of four hours for the discovery phase and one hour for measuring each of the quality measures. We ran the second evaluation on a 6-core Intel Xeon CPU E5-2699 v4 @ 2.20GHz with 128GB RAM running Java 8, and we increased the heap and stack spaces to 25GB and 15GB respectively, using a timeout of 24 hours for each APDA-log evaluation.

4.4. Benchmark Results

The results of the default parameters evaluation are shown in Tables 6, 7, 8, and 9. In the tables, we used "-" to report that a given accuracy or complexity measurement could not be reliably obtained due to syntactical or behavioral issues in the

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity			Sound	Exec. Time(s)
		Fitness	Precision	F-score		Size	CFC	Struct.		
BPIC12	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	**0.98**	0.50	**0.66**	**0.98**	**59**	37	**1.00**	yes	**6.60**
	ETM	0.44	**0.82**	0.57	t/o	67	**16**	**1.00**	yes	14,400
	FO	-	-	-	-	102	117	0.13	no	9.66
	S-HM$_6$	-	-	-	-	88	46	0.40	no	227.80
	HILP	-	-	-	-	300	460	-	no	772.20
BPIC13$_{\text{cp}}$	$\alpha\$$	-	-	-	-	18	9	-	no	10,112.60
	IM	0.82	**1.00**	0.90	0.82	**9**	4	**1.00**	yes	0.10
	ETM	**1.00**	0.70	0.82	t/o	38	38	**1.00**	yes	14,400
	FO	-	-	-	-	25	23	0.60	no	**0.06**
	S-HM$_6$	0.94	0.99	**0.97**	**0.94**	15	6	**1.00**	yes	130.0
	HILP	-	-	-	-	10	**3**	-	yes	0.10
BPIC13$_{\text{inc}}$	$\alpha\$$	0.35	0.91	0.51	t/o	15	7	0.47	yes	4,243.14
	IM	0.92	0.54	0.68	**0.92**	13	7	**1.00**	yes	1.00
	ETM	**1.00**	0.51	0.68	t/o	32	144	**1.00**	yes	14,400
	FO	-	-	-	-	43	54	0.77	no	1.41
	S-HM$_6$	0.91	**0.96**	**0.93**	0.91	**9**	**4**	**1.00**	yes	**0.80**
	HILP	-	-	-	-	24	9	-	yes	2.50
BPIC14$_{\text{f}}$	$\alpha\$$	0.47	0.63	0.54	t/o	62	36	0.31	yes	14,057.48
	IM	**0.89**	0.64	0.74	**0.89**	31	18	**1.00**	yes	**3.40**
	ETM	0.61	**1.00**	**0.76**	t/o	**23**	**9**	**1.00**	yes	14,400
	FO	-	-	-	-	37	46	0.38	no	27.73
	S-HM$_6$	-	-	-	-	202	132	0.73	no	147.40
	HILP	-	-	-	-	80	59	-	no	7.30
BPIC15$_{\text{1f}}$	$\alpha\$$	0.71	0.76	0.73	t/o	219	91	0.22	yes	3,545.9
	IM	0.97	0.57	0.71	**0.96**	164	108	**1.00**	yes	**0.60**
	ETM	0.56	**0.94**	0.70	t/o	**67**	**19**	**1.00**	yes	14,400
	FO	**1.00**	0.76	**0.87**	0.94	146	91	0.25	yes	1.02
	S-HM$_6$	-	-	-	-	204	116	0.56	no	128.10
	HILP	-	-	-	-	282	322	-	no	4.40
BPIC15$_{\text{2f}}$	$\alpha\$$	-	-	-	-	348	164	0.08	no	8,787.48
	IM	0.93	0.56	0.70	0.94	193	123	**1.00**	yes	0.70
	ETM	0.62	**0.91**	**0.74**	t/o	**95**	**32**	**1.00**	yes	14,400
	FO	-	-	-	-	195	159	0.09	no	**0.61**
	S-HM$_6$	**0.98**	0.59	**0.74**	**0.97**	259	150	0.29	yes	163.2
	HILP	-	-	-	-	-	-	-	-	t/o
BPIC15$_{\text{3f}}$	$\alpha\$$	-	-	-	-	319	169	0.03	no	10,118.15
	IM	**0.95**	0.55	0.70	**0.95**	159	108	**1.00**	yes	1.30
	ETM	0.68	**0.88**	0.76	t/o	**84**	**29**	**1.00**	yes	14,400
	FO	-	-	-	-	174	164	0.06	no	**0.89**
	S-HM$_6$	**0.95**	0.67	**0.79**	**0.95**	159	151	0.13	yes	139.90
	HILP	-	-	-	-	433	829	-	no	1,062.90
BPIC15$_{\text{4f}}$	$\alpha\$$	-	-	-	-	272	128	0.13	no	6,410.25
	IM	0.96	0.58	0.73	0.96	162	111	**1.00**	yes	0.7
	ETM	0.65	**0.93**	0.77	t/o	**83**	**28**	**1.00**	yes	14,400
	FO	-	-	-	-	157	127	0.14	no	**0.50**
	S-HM$_6$	**0.99**	0.64	**0.78**	**0.99**	209	137	0.37	yes	136.90
	HILP	-	-	-	-	364	593	-	no	14.7
BPIC15$_{\text{5f}}$	$\alpha\$$	0.62	0.75	0.68	t/o	280	126	0.10	yes	7,603.19
	IM	0.94	0.18	0.30	0.94	134	95	**1.00**	yes	1.50
	ETM	0.57	**0.94**	0.71	t/o	**88**	**18**	**1.00**	yes	14,400
	FO	**1.00**	0.71	**0.83**	**1.00**	166	125	0.15	yes	**0.56**
	S-HM$_6$	**1.00**	0.70	0.82	**1.00**	211	135	0.35	yes	141.90
	HILP	-	-	-	-	-	-	-	-	t/o
BPIC17$_{\text{f}}$	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	**0.98**	0.70	0.82	**0.98**	**35**	20	**1.00**	yes	**13.30**
	ETM	0.76	**1.00**	**0.86**	t/o	42	**4**	**1.00**	yes	14,400
	FO	-	-	-	-	98	82	0.25	no	64.33
	S-HM$_6$	0.95	0.62	0.75	0.94	42	13	0.97	yes	143.20
	HILP	-	-	-	-	222	330	-	no	384.50

Table 6: Default parameters evaluation results for the BPIC logs.

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity			Sound?	Exec. Time (sec)
		Fitness	Precision	F-score		Size	CFC	Struct.		
RTFMP	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	0.99	0.70	0.82	**0.99**	34	**20**	**1.00**	yes	10.90
	ETM	0.99	0.92	0.95	t/o	57	32	**1.00**	yes	14,400
	FO	**1.00**	0.94	**0.97**	0.97	**31**	32	0.19	yes	**2.57**
	S-HM$_6$	0.98	**0.95**	0.96	0.98	163	97	**1.00**	yes	262.70
	HILP	-	-	-	-	57	53	-	no	3.50
SEPSIS	$\alpha\$$	-	-	-	-	146	156	0.01	no	3,883.12
	IM	**0.99**	0.45	0.62	**0.96**	**50**	**32**	**1.00**	yes	0.40
	ETM	0.83	**0.66**	**0.74**	t/o	108	101	**1.00**	yes	14,400
	FO	-	-	-	-	60	63	0.28	no	**0.17**
	S-HM$_6$	0.92	0.42	0.58	0.92	279	198	**1.00**	yes	242.70
	HILP	-	-	-	-	87	129	-	no	1.60

Table 7: Default parameters evaluation results for the public logs.

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity			Sound	Exec. Time(s)
		Fitness	Precision	F-score		Size	CFC	Struct.		
PRT1	$\alpha\$$	-	-	-	t/o	45	34	-	no	11,168.54
	IM	0.90	0.67	0.77	**0.90**	**20**	**9**	**1.00**	yes	2.08
	ETM	**0.99**	**0.81**	**0.89**	t/o	23	12	**1.00**	yes	14,400
	FO	-	-	-	-	30	28	0.53	no	**0.95**
	S-HM$_6$	0.88	0.77	0.82	0.88	59	39	**1.00**	yes	122.16
	HILP	-	-	-	-	195	271	-	no	2.59
PRT2	$\alpha\$$	-	-	-	-	134	113	0.25	no	3,438.72
	IM	ex	ex	ex	ex	**45**	33	**1.00**	yes	1.41
	ETM	**0.57**	**0.94**	**0.71**	t/o	86	**21**	**1.00**	yes	14,400
	FO	-	-	-	-	76	74	0.59	no	**0.88**
	S-HM$_6$	-	-	-	-	67	105	0.43	no	1.77
	HILP	-	-	-	-	190	299	-	no	21.33
PRT3	$\alpha\$$	0.67	0.76	0.71	0.67	70	40	0.11	yes	220.11
	IM	0.98	0.68	0.80	0.98	37	**20**	**1.00**	yes	**0.44**
	ETM	0.98	**0.86**	**0.92**	t/o	51	37	**1.00**	yes	14,400
	FO	**1.00**	**0.86**	**0.92**	**1.00**	**34**	37	0.32	yes	0.50
	S-HM$_6$	**1.00**	0.83	0.91	**1.00**	40	38	0.43	yes	0.67
	HILP	-	-	-	-	343	525	-	no	0.73
PRT4	$\alpha\$$	0.86	**0.93**	0.90	t/o	**21**	**10**	**1.00**	yes	13,586.48
	IM	0.93	0.75	0.83	0.93	27	13	**1.00**	yes	**1.33**
	ETM	0.84	0.85	0.84	t/o	64	28	**1.00**	yes	14,400
	FO	-	-	-	-	37	40	0.54	no	6.33
	S-HM$_6$	**1.00**	0.86	**0.93**	**1.00**	370	274	**1.00**	yes	241.57
	HILP	-	-	-	-	213	306	-	no	5.31
PRT5	$\alpha\$$	1.00	1.00	1.00	1.00	10	1	1.00	yes	2.02
	IM	1.00	1.00	1.00	1.00	10	1	1.00	yes	0.03
	ETM	1.00	1.00	1.00	1.00	10	1	1.00	yes	2.49
	FO	1.00	1.00	1.00	1.00	10	1	1.00	yes	**0.02**
	S-HM$_6$	1.00	1.00	1.00	1.00	10	1	1.00	yes	0.11
	HILP	1.00	1.00	1.00	1.00	10	1	1.00	yes	0.05
PRT6	$\alpha\$$	0.80	0.77	0.79	0.80	38	17	0.24	yes	40.10
	IM	0.99	0.82	0.90	0.99	23	**10**	**1.00**	yes	2.30
	ETM	0.98	0.80	0.88	t/o	41	16	**1.00**	yes	14,400
	FO	**1.00**	**0.91**	**0.95**	**1.00**	**22**	17	0.41	yes	**0.05**
	S-HM$_6$	**1.00**	**0.91**	**0.95**	**1.00**	**22**	17	0.41	yes	0.42
	HILP	-	-	-	-	157	214	-	no	0.13

Table 8: Default parameters evaluation results for the proprietary logs - Part 1/2.

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity			Sound?	Exec. Time (sec)
		Fitness	Precision	F-score		Size	CFC	Struct.		
PRT7	α\$	0.85	0.90	0.88	0.85	29	**9**	0.48	yes	143.66
	IM	**1.00**	0.73	0.84	**1.00**	29	13	**1.00**	yes	0.13
	ETM	0.90	0.81	0.85	t/o	60	29	**1.00**	yes	14,400
	FO	0.99	**1.00**	0.99	0.99	**26**	16	0.39	yes	**0.08**
	S-HM$_6$	**1.00**	**1.00**	**1.00**	**1.00**	163	76	**1.00**	yes	249.74
	HILP	-	-	-	-	278	355	-	no	0.27
PRT8	α\$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	**0.98**	0.33	0.49	**0.93**	111	92	**1.00**	yes	**0.41**
	ETM	0.35	**0.88**	**0.50**	t/o	**75**	**12**	**1.00**	yes	14,400
	FO	-	-	-	-	228	179	0.74	no	0.55
	S-HM$_6$	-	-	-	-	388	323	0.87	no	370.66
	HILP	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
PRT9	α\$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	0.90	0.61	0.73	0.89	28	16	**1.00**	yes	63.70
	ETM	0.75	0.49	0.59	0.74	**27**	**13**	**1.00**	yes	1,266.71
	FO	-	-	-	-	32	45	0.72	no	**42.83**
	S-HM$_6$	**0.96**	**0.98**	**0.97**	**0.96**	723	558	**1.00**	yes	318.69
	HILP	-	-	-	-	164	257	-	no	51.47
PRT10	α\$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	0.96	0.79	0.87	0.96	**41**	**29**	**1.00**	yes	2.50
	ETM	**1.00**	0.63	0.77	t/o	61	45	**1.00**	yes	14,400
	FO	0.99	**0.93**	**0.96**	**0.99**	52	85	0.64	yes	**0.98**
	S-HM$_6$	-	-	-	-	77	110	-	no	1.81
	HILP	-	-	-	-	846	3130	-	no	2.55
PRT11	α\$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	t/o	t/o	t/o	t/o	549	365	**1.00**	yes	121.50
	ETM	**0.10**	**1.00**	**0.18**	t/o	**21**	**3**	**1.00**	yes	14,400
	FO	-	-	-	-	680	713	0.68	no	**81.33**
	S-HM$_6$	ex	ex	ex	ex	ex	ex	ex	ex	ex
	HILP	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
PRT12	α\$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	**1.00**	0.77	**0.87**	**1.00**	32	25	**1.00**	yes	3.94
	ETM	0.63	**1.00**	0.77	t/o	**21**	**8**	**1.00**	yes	14,400
	FO	-	-	-	-	87	129	0.38	no	**1.67**
	S-HM$_6$	-	-	-	-	4370	3191	**1.00**	yes	347.57
	HILP	-	-	-	-	926	2492	-	no	7.34

Table 9: Default parameters evaluation results for the proprietary logs - Part 2/2.

Log	**Discovery Method**	**Accuracy**			**Gen.**	**Complexity**			**Exec.**
		Fitness	**Precision**	**F-score**	**(3-Fold)**	**Size**	**CFC**	**Struct.**	**Time (sec)**
Frequency Absolute Best	$\alpha\$$	0	1	0	0	1	2	1	0
	IM	9	1	2	**12**	7	5	**24**	7
	ETM	5	**15**	**9**	0	**10**	**13**	**24**	0
	FO	5	4	6	4	4	0	0	**16**
	S-HM$_6$	**10**	5	8	10	2	1	9	0
	HILP	0	0	0	0	0	1	0	1
Frequency Second Best	$\alpha\$$	0	4	2	0	1	2	1	0
	IM	**12**	**9**	**10**	**9**	**14**	**15**	0	**13**
	ETM	6	0	5	0	3	5	0	0
	FO	2	2	1	2	4	4	**12**	5
	S-HM$_6$	3	7	5	4	3	1	11	1
	HILP	0	0	0	0	1	0	0	6
Total	$\alpha\$$	0	5	2	0	2	4	2	0
	IM	**21**	10	12	**21**	**21**	**20**	**24**	20
	ETM	11	**15**	**14**	0	13	18	**24**	0
	FO	7	6	7	6	8	4	12	**21**
	S-HM$_6$	13	12	13	14	5	2	20	1
	HILP	0	0	0	0	1	1	0	7

Table 10: Best score frequencies for each quality dimension (default parameters evaluation).

discovered model (i.e., a disconnected model or an unsound model). In particular, a "-" in generalization means that one or more of the 3-folds did not yield a reliable fitness result. Additionally, to report the occurrence of a timeout or an exception during the execution of an APDA we used "*t/o*" and "*ex*", respectively. We highlighted in bold the best score for each measure on each log, we underlined the second-best score, and we summarized these achievements in Table 10.

The first evaluation shows the absence of a clear winner among the APDAs tested, although almost each of them clearly showed specific benefits and limitations.

HILP experienced severe difficulties in producing useful outputs, regardless of the input event log (except the case of PRT5). Over all, HILP performance was consistent throughout the dataset, producing disconnected models or models containing multiple end places without providing information about the final marking (a well defined final marking is required in order to measure fitness and precision). Due to these difficulties, we could only assess model complexity for HILP, except for the simplest event log (the *PRT5*), where HILP had performance comparable to the other methods.

$\alpha\$$ suffered of scalability issues, timing out in eight event logs (33% of the times) and being the approach with the highest rate of time outs. It is not clear the cause of this issue, however, we record that $\alpha\$$ is the second slowest approach, behind ETM. While for the latter, the lack of efficiency is counterbalanced by the quality of the discovered models, the models discovered by $\alpha\$$ do not stand out in accuracy or in complexity. Nevertheless, $\alpha\$$ produced models striking a good balance between fitness and precision (except for the $BPIC13_{inc}$ log).

FO struggled to deliver sound models, discovering one sound model out of three. This makes FO unreliable when it comes to discovering sound models, however, when it does discover a sound model, it is usually highly fitting. Indeed, FO scored the best fitness five times out of 24, or five times out of eight, if we consider only the subset of sound models discovered by FO.

S-HM$_6$ performed better than FO, even though its discovered models were also often unsound. Out of the 16 sound models discovered, ten scored the best fitness and generalization, making S-HM$_6$ one of the best APDAs for these two quality dimensions along with IM (see Table 10). Precision varied according to the input event log, demonstrating that the performance of S-HM$_6$ is bound to the type of input log. Whilst it was not the absolute best in F-score, S-HM$_6$ achieved most of the times good results, placing itself as the second best APDA for F-score.

The remaining two APDAs, namely IM and ETM, consistently performed very well across the whole evaluation, excelling either in fitness, precision, F-score or generalization, and simultaneously striking the highest simplicity for the discovered process models. IM scored 20 times a fitness greater than 0.90 (of which 9 times the highest), and it achieved similar results for generalization. Despite this, IM did not stand out for its precision, nor for its F-score. ETM, instead, achieved respectively 19 times a precision greater than 0.80, and it was the best 15 times. However, ETM scored high precision at the cost of lower fitness, this can be due to the design of ETM. We remind that ETM is a genetic algorithm, it starts from a very simple model (including little behaviour from the log) and it improves the model (adding new behaviour) slowly at each generation. ETM performed well also with its F-score (along with S-HM$_6$), achieving an F-score above 0.80 9 times out of 24, outperforming the other APDAs.

In terms of complexity, IM and ETM stood out among all the APDAs (see Table 10). IM and ETM always discovered sound and fully block-structured models (structuredness equal to 1.00). ETM and IM discovered the smallest or second-smallest model for more than 50% and 80% of the logs respectively. These models also had low CFC, and were the ones with the lowest CFC on 13 logs (ETM) and on five logs (IM). On the execution time, FO was the clear winner, followed by IM. The former was the fastest discovery approach 16 times out of 24, discovering a model in less than a second for 13 logs. In contrast, ETM was the slowest APDA, reaching the timeout of four hours for 22 logs.

The results of the hyper-parameter optimization evaluation are shown in Tables 11–14. Here we marked with a "*" the discovery approaches that were not able to complete the exploration of the solution space within 24 hours of timeout time. The purpose of this second evaluation was to understand if the APDAs can achieve higher F-score when optimally tuned, and what price they pay for such an improvement, i.e. at the cost of which other quality dimension. In line with our goal, Tables 11 and 12 report the accuracy and complexity scores of the discovered models with the highest F-score. We note that some of the insights gained from the default parameters evaluation do not hold anymore. FO and S-HM$_6$ were almost always able to discover sound models from each log for at least one input configuration. FO outperformed IM in fitness, by scoring the best fitness 15 times.

Log	**Discovery Method**	**Accuracy**			**Gen. (3-Fold)**	**Complexity**		
		Fitness	**Precision**	**F-score**		**Size**	**CFC**	**Struct.**
BPIC12	IM	0.90	**0.69**	0.78	0.91	**69**	**46**	**1.00**
	FO*	**1.00**	0.07	0.14	-	112	1369	**1.00**
	S-HM$_6$	0.96	0.67	**0.79**	**0.96**	97	110	0.63
BPIC13$_{\text{cp}}$	IM	**0.99**	**0.98**	**0.98**	**0.99**	**11**	**5**	**1.00**
	FO	0.00	0.42	0.00	-	19	16	**1.00**
	S-HM$_6$	0.96	0.92	0.94	0.96	20	14	0.80
BPIC13$_{\text{inc}}$	IM	0.90	0.87	0.89	0.90	**13**	**5**	**1.00**
	FO	0.00	0.36	0.00	-	42	76	0.88
	S-HM$_6$	**0.93**	**0.98**	**0.96**	**0.93**	16	10	**1.00**
BPIC14$_{\text{f}}$	IM	0.75	**0.97**	0.85	0.75	**19**	**4**	**1.00**
	FO	**0.97**	0.81	**0.88**	0.31	27	34	0.56
	S-HM$_6$	0.91	0.84	**0.88**	**0.91**	178	117	0.97
BPIC15$_{\text{1f}}$	IM	0.81	0.68	0.74	0.83	**140**	**70**	**1.00**
	FO	**1.00**	0.76	0.87	**0.94**	146	91	0.26
	S-HM$_6$	0.88	**0.89**	**0.89**	0.58	1576	550	**1.00**
BPIC15$_{\text{2f}}$	IM	0.71	**0.76**	0.74	0.69	**141**	**61**	**1.00**
	FO	**0.99**	0.63	**0.77**	**0.99**	195	164	0.09
	S-HM$_6$	**0.99**	0.62	0.76	**0.99**	246	167	0.19
BPIC15$_{\text{3f}}$	IM	0.65	**0.99**	**0.79**	0.63	**73**	**8**	**1.00**
	FO	**0.99**	0.60	0.75	**0.99**	162	163	0.07
	S-HM$_6$	0.81	0.77	**0.79**	0.81	231	77	0.97
BPIC15$_{\text{4f}}$	IM	0.73	**0.84**	0.78	0.75	**108**	**42**	**1.00**
	FO	**1.00**	0.67	**0.80**	**1.00**	155	128	0.14
	S-HM$_6$	0.99	0.66	0.79	0.99	217	145	0.36
BPIC15$_{\text{5f}}$	IM	0.64	0.88	0.74	0.65	**105**	**34**	**1.00**
	FO	**1.00**	0.71	0.83	**1.00**	166	125	0.15
	S-HM$_6$	0.82	**0.94**	**0.87**	0.81	610	166	0.96
BPIC17$_{\text{f}}$	IM	**1.00**	**0.70**	**0.82**	**1.00**	**39**	**24**	**1.00**
	FO*	-	-	-	-	-	-	-
	S-HM$_6$	0.97	**0.70**	0.81	0.97	51	25	**1.00**
RTFMP	IM	0.94	0.98	0.96	0.94	**28**	**10**	**1.00**
	FO	**1.00**	0.94	**0.97**	0.84	31	32	0.19
	S-HM$_6$	0.95	**0.99**	**0.97**	**0.95**	82	30	**1.00**
SEPSIS	IM	0.62	**0.98**	**0.76**	0.76	**31**	**14**	**1.00**
	FO	**0.96**	0.36	0.53	0.30	51	109	0.33
	S-HM$_6$	0.80	0.39	0.52	**0.86**	299	187	**1.00**

Table 11: Scores of the models with the best F-score discovered with hyper-parameter optimization (public logs).

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity		
		Fitness	Precision	F-score		Size	CFC	Struct.
PRT1	IM	0.91	0.89	0.90	0.91	**24**	**11**	**1.00**
	FO	**0.98**	0.92	0.95	**0.99**	25	29	0.72
	S-HM$_6$	0.95	**0.97**	**0.96**	0.95	37	29	0.92
PRT2	IM	-	-	-	-	-	-	-
	FO	**1.00**	**0.17**	**0.30**	**1.00**	**55**	**241**	**0.93**
	S-HM$_6$	-	-	-	-	-	-	-
PRT3	IM	0.87	**0.93**	0.90	0.87	**27**	**8**	**1.00**
	FO	**1.00**	0.86	**0.92**	**1.00**	34	37	0.32
	S-HM$_6$	0.99	0.85	0.91	0.96	40	34	0.48
PRT4	IM	0.86	**1.00**	0.92	0.86	**21**	**5**	**1.00**
	FO	**1.00**	0.87	0.93	-	32	41	0.50
	S-HM$_6$	0.93	0.96	**0.95**	**0.93**	66	55	0.77
PRT5	IM	1.00	1.00	1.00	**1.00**	12	1	**1.00**
	FO	1.00	1.00	1.00	0.95	**10**	1	**1.00**
	S-HM$_6$	1.00	1.00	1.00	**1.00**	12	1	**1.00**
PRT6	IM	0.90	**1.00**	0.95	0.90	**17**	**2**	**1.00**
	FO	**1.00**	0.91	0.95	0.96	22	17	0.41
	S-HM$_6$	0.98	0.96	**0.97**	**0.98**	24	15	0.46
PRT7	IM	0.88	**1.00**	0.93	0.88	**23**	**5**	**1.00**
	FO	0.99	**1.00**	0.99	0.99	26	16	0.39
	S-HM$_6$	**1.00**	**1.00**	**1.00**	**1.00**	165	76	**1.00**
PRT8	IM*	**1.00**	0.09	0.16	**0.99**	**95**	**86**	**1.00**
	FO	-	-	-	-	-	-	-
	S-HM$_6$	0.93	**0.42**	**0.58**	0.89	221	422	0.83
PRT9	IM	0.93	0.71	0.80	0.93	**28**	**14**	**1.00**
	FO	-	-	-	-	-	-	-
	S-HM$_6$	**0.99**	**0.99**	**0.99**	**0.99**	41	59	0.68
PRT10	IM	**1.00**	0.81	0.89	**1.00**	**47**	**33**	**1.00**
	FO	0.99	**0.93**	**0.96**	-	52	85	0.64
	S-HM$_6$	0.98	0.83	0.90	0.98	1440	972	**1.00**
PRT12	IM	0.93	**0.92**	**0.93**	0.93	**37**	**26**	**1.00**
	FO	**1.00**	0.80	0.89	**0.94**	60	237	0.87
	S-HM$_6$	0.88	0.67	0.76	0.88	3943	2314	**1.00**

Table 12: Scores of the models with the best F-score discovered with hyper-parameter optimization (proprietary logs).

Metric	Discovery Method	BPIC Logs										RTFMP	SEPSIS
		12	**13_{cp}**	**13_{inc}**	**14_f**	**15_{1f}**	**15_{2f}**	**15_{3f}**	**15_{4f}**	**15_{5f}**	**17_f**		
Fitness	IM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	**1.00**	0.00	0.00	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	-	**1.00**	**1.00**
	S-HM$_6$	0.96	**1.00**	0.93	**1.00**	**1.00**	0.99	0.99	**1.00**	**1.00**	**1.00**	0.95	0.80
Prec.	IM	**0.92**	**1.00**	0.89	**1.00**	**0.97**	**1.00**	**0.99**	**0.91**	**0.99**	**0.89**	0.98	**0.98**
	FO	0.07	0.42	0.36	0.81	0.76	0.63	0.60	0.67	0.71	-	0.94	0.36
	S-HM$_6$	0.67	0.92	**0.98**	0.84	0.89	0.78	0.77	0.66	0.94	0.70	**0.99**	0.39
F-score	IM	0.78	**0.98**	0.89	0.85	0.74	0.74	**0.79**	0.78	0.74	**0.82**	0.96	**0.76**
	FO	0.14	0.00	0.00	**0.88**	0.87	**0.77**	0.75	**0.80**	0.83	-	**0.97**	0.53
	S-HM$_6$	**0.79**	0.94	**0.96**	**0.88**	**0.89**	0.76	**0.79**	0.79	**0.87**	0.81	**0.97**	0.52
Gen. (3-Fold)	IM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	-	-	0.00	0.96	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	-	0.90	0.94
	S-HM$_6$	0.96	0.99	0.93	**1.00**	**1.00**	0.99	0.99	**1.00**	**1.00**	**1.00**	0.95	0.86
Size	IM	**15**	9	**9**	17	**63**	**30**	**17**	**25**	**32**	23	**11**	**14**
	FO	112	19	39	27	113	137	114	111	117	-	23	40
	S-HM$_6$	87	**8**	16	**13**	74	246	207	139	232	**22**	82	299
CFC	IM	**8**	2	**3**	2	10	**7**	**8**	**9**	**9**	6	**5**	**9**
	FO	1369	16	54	34	47	57	53	46	44	-	13	35
	S-HM$_6$	65	**0**	10	**0**	**0**	167	77	35	140	**0**	30	187
Struct.	IM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.29	**1.00**	**1.00**	**1.00**	-	**1.00**	**1.00**
	S-HM$_6$	0.77	**1.00**	**1.00**	0.97	**1.00**	0.99	0.98	0.37	0.96	**1.00**	**1.00**	**1.00**

Table 13: Best scores achieved in hyper-parameter evaluation by each approach on each quality dimension (public logs).

Metric	Discovery Method	Proprietary (PRT) Logs										
		# 1	**# 2**	**# 3**	**# 4**	**# 5**	**# 6**	**# 7**	**# 8**	**# 9**	**# 10**	**# 12**
Fitness	IM	**1.00**	-	**1.00**	**1.00**	1.00	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**1.00**	**1.00**	-	-	**1.00**	**1.00**
	S-HM$_6$	**1.00**	-	**1.00**	**1.00**	1.00	**1.00**	**1.00**	**1.00**	0.99	**1.00**	0.88
Prec.	IM	0.89	-	**0.93**	**1.00**	1.00	**1.00**	**1.00**	0.09	0.89	0.95	**0.99**
	FO	0.92	**0.17**	0.89	0.94	1.00	0.91	**1.00**	-	-	**0.96**	0.80
	S-HM$_6$	**0.97**	-	0.85	0.96	1.00	0.96	**1.00**	**0.42**	**0.99**	0.83	0.67
F-score	IM	0.90	-	0.90	0.92	1.00	0.95	0.93	0.16	0.80	0.89	**0.93**
	FO	0.95	**0.30**	**0.92**	0.93	1.00	0.95	0.99	-	-	**0.96**	0.89
	S-HM$_6$	**0.96**	-	0.91	**0.95**	1.00	**0.97**	**1.00**	**0.58**	**0.99**	0.90	0.76
Gen. (3-Fold)	IM	**1.00**	-	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**
	FO	**1.00**	**1.00**	**1.00**	-	0.95	0.96	**1.00**	-	-	-	0.94
	S-HM$_6$	**1.00**	-	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.98	0.99	**1.00**	0.88
Size	IM	14	-	**27**	**21**	12	**17**	23	95	**16**	27	**25**
	FO	22	**55**	30	30	**10**	22	**21**	-	-	48	60
	S-HM$_6$	**13**	-	37	66	12	24	40	**59**	41	**23**	3943
CFC	IM	4	-	**8**	**5**	**1**	**2**	**5**	86	**2**	10	**18**
	FO	16	**74**	24	33	**1**	17	7	-	-	44	66
	S-HM$_6$	**0**	-	27	55	**1**	15	15	**0**	58	**0**	2314
Struct.	IM	**1.00**	-	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	0.77	**0.93**	0.57	0.54	**1.00**	0.41	0.81	-	-	0.81	0.87
	S-HM$_6$	**1.00**	-	0.72	**1.00**	**1.00**	0.50	**1.00**	0.85	**1.00**	**1.00**	**1.00**

Table 14: Best scores achieved in hyper-parameter evaluation by each approach on each quality dimension (proprietary logs).

IM performed better in precision (being 13 times the best). Further, IM delivered the simplest model for size, CFC and structuredness 20 times. S-HM$_6$ turned to be the most balanced discovery approach, scoring the highest F-score most of the times (13 out of 23), at the cost of complexity, indeed, S-HM$_6$ was never the best for size or CFC. Only the results for generalization got mixed, with S-HM$_6$ scoring the best generalization 11 times, followed very closely by FO (nine times) and by IM (five times).

Finally, Tables 13 and 14 report the best score that each discovery approach can achieve in each dimension. FO and IM are always able to maximally optimize fitness, scoring always 1.00 as their best score. S-HM$_6$ performs slightly worse in fitness, being always in the range 0.90-1.00. Similar are the results for generalization, with FO and IM leading again, and S-HM$_6$ following closely. As for precision, FO and S-HM$_6$ get over 0.80 66% of the times only, whilst IM strikes better results, being always in the range 0.90-1.00 (excluding the two outlier models from the logs PRT2 and PRT8). Finally, F-score results reveal the ability of the discovery approaches to properly balance fitness and precision. We note that IM lacks such an ability, reaching an F-score above 0.90 only 30% of the times, with values below 0.80 35% of the times. These results are as such despite IM can reach very high values both in fitness and in precision, individually. FO follows IM as worst performer in F-score with similar outcomes, whilst S-HM$_6$ distinguished itself with scores in the range 0.80-1.00 and often over 0.90 (more than 50% of the times). As for model simplicity, IM leads the way. Whilst FO and S-HM$_6$ struggle to optimize both size and CFC.

In conclusion, an approach outperforming all others across all measures could not be identified. Despite this, when it comes to default parameters IM and ETM showed to be the most effective approaches when the focus is either on fitness, precision, or F-score, and these APDAs all yield simple process models. However, even these two methods suffer from a common weakness, which is their inability to handle large-scale real-life logs, as reported for the PRT11 log in our evaluation. On the other hand, the hyper-parameter optimization exercise showed that also FO and S-HM$_6$ can perform well, though at the expenses of long execution times (up to 24 hours for some logs) and powerful computational resources.

The complete results of the hyper-parameter optimization evaluation are available online[11] and include excel sheets reporting for each APDA (IM, FO, SM, and S-HM$_6$) the results obtained for each input configuration on each event log tested.

4.5. Threats to Validity

The experimental evaluation is limited in scope to techniques that produce Petri nets (or models in languages such as BPMN or Process Trees, which can be di-

---

[11] At `https://doi.org/10.5281/zenodo.1219321`

rectly translated to Petri nets). Also, it only considers main studies identified in our SLR (reported in Chapter 2) with an accessible implementation. In order to compensate for these shortcomings, we published the benchmark framework as open-source software in order to enable researchers both to reproduce the results herein reported and to run the same evaluation for other APDAs, or for alternative configurations of the evaluated APDAs.

Another limitation derives from the measurements of fitness, precision, and generalization, which assume that the input event log is noise-free. We remark that it is not possible to determine whether an event log contains behavioural inconsistencies (i.e. noise) without having a reference model to compare with, and even more complex would be to detect such inconsistencies. However, even though the event logs are not noise-free, any possible measurement error would not discriminate a single APDA, but rather affect equally all of them as a systematic bias.

The precision measure we used in our benchmark [129] does not totally adhere to the original definition of precision as given in Section 3.1. This may pose a threat to the interpretation of the results. On the other hand, all the state-of-the-art precision measures [4, 5] have their drawbacks and flaws, and none of those applicable to real-life event logs operationalizes transparently and genuinely the original definition of precision. Therefore, our choice of precision measure should be seen as a design choice with its inherited limitations.

The random selection of the folds of each log when computing the 3-fold generalization should be considered as a limitation as well. We did check the fitness values obtained for the three folds of each algorithm, and found variations in these values ranging from 1 to 3 percentage points, suggesting that the algorithms did produce different results for different folds. Nevertheless, given that the folds were generated randomly, we have no guarantee that there is sufficient difference between the folds. To attenuate this threat, we selected a small number of folds in order to ensure that the test-fold would contain additional traces with respect to the union of the training folds. Indeed, in our context, the higher the number of folds the lower is the chance that the test-fold (1 out of K) contains traces not observed in the training folds (K-1 out of K).

Finally, one could argue that the use of 24 event logs limits to a certain extend the generalizability of the conclusions. However, the event logs included in the evaluation are all real-life logs of different sizes and features, including different application domains. Also, to the best of our knowledge and according to the studies analysed in Chapter 3, our benchmark dataset is so far the largest real-life dataset employed to assess APDAs. To mitigate this limitation, we have structured the released benchmark framework in such a way that the benchmark can be seamlessly rerun with additional datasets.

4.6. Related Work

A previous survey and benchmark of automated process discovery methods has been reported by De Weerdt et al. [3]. This survey covered 27 approaches, and it assessed seven of them. We used it as starting point for our study.

The benchmark reported by De Weerdt et al. [3] includes seven approaches, namely AGNEsMiner, $\alpha+$, $\alpha{+}{+}$, Genetic Miner (and a variant thereof), Heuristics Miner and ILP Miner. In comparison, our benchmark includes $\alpha\$$ (which is an improved version of $\alpha+$ and $\alpha{+}{+}$), Structured Heuristics Miner (which is an extension of Heuristics Miner), Hybrid ILP Miner (an improvement of ILP), Evolutionary Tree Miner (which is a genetic algorithm postdating the evaluation of De Weerdt et al. [3]). Notably, we did not include AGNEsMiner due to the very long execution times (as suggested by the authors in a conversation over emails exchanged during this work).

Another difference with respect to the previous survey [3], is that in our study we based our evaluation both on public and proprietary real-life event logs, whilst the evaluation of De Weerdt et al. [3] is solely based on artificial event logs and closed datasets, due to the unavailability of public datasets at the time of that study.

In terms of results, De Weerdt et al. [3] found that Heuristics Miner achieved a better F-score than other approaches and generally produced simpler models, while ILP achieved the best fitness at the expense of low precision and high model complexity. Our results show that ETM and IM excelled for precision and fitness (respectively), while at the same time discovering simple process models.

Another previous survey in the field is outdated [138] and a more recent one is not intended to be comprehensive [139], but rather limits on plug-ins available in the ProM toolset. Finally a related effort is CoBeFra – a tool suite for measuring fitness, precision and model complexity of automatically discovered process models [140], however it does not providing a set of event logs to use as benchmark.

4.7. Summary

In this chapter we addressed the second part of our **RQ1**: *what are the state-of-the-art APDAs' strengths and limitations?*

To assess the strengths and the limitations of the state-of-the-art APDAs, we designed and implemented a benchmark framework which allows reproducible evaluations of APDAs on a set of twelve publicly available real-life event logs. Our benchmark framework integrates all the existing and most important evaluation measures: fitness, precision, F-score, generalization, complexity, soundness, and execution time (for assessing scalability). However, the accuracy and complexity measures were designed only for procedural process models. Such a limitation can be seen as a limitation of the existing measurements or as a limitation of the APDAs that output declarative process models. As a consequence, we were forced to restrict our evaluation to APDAs that output procedural process models, precisely, Petri nets and any type of process models that can be converted into a Petri nets in a deterministic manner (e.g. BPMN models, process trees). We decided to focus our evaluation and comparison on the procedural APDAs because they have a higher practical relevance than their declarative counterparts, and because of the lack of accuracy measures that allow to assess declarative process models and compare their quality against the procedural ones. On top of this, we note that the great majority of the state-of-the-art APDAs do not have a working or accessible implementation, this hampers their systematic evaluation, so that we can only rely on the results reported in the respective papers.

The results of our benchmark give an overview of the benefits of the procedural APDAs, as well as their limitations. The latter include: (i) strong differences across the various quality measures in the output models; (ii) lack of scalability for large logs; (iii) difficulties in discovering sound process models. The majority of the assessed APDAs are not able to systematically excel in accuracy (fitness, precision, F-score, and generalization at the same time), nor they can guarantee to discover sound process models. The exceptions were only IM and ETM, who clearly represent the state of the art in automated process discovery. IM and ETM proved to be above the average, indeed, these two approaches were the only to consistently perform very well in at least one quality dimension, respectively, fitness (IM) and precision (ETM). As a by-product of their quality, often IM or ETM could achieve also the best F-scores among the other APDAs. Nevertheless, IM and ETM rarely achieved outstanding F-score values, highlighting their inability to efficiently balance fitness and precision. Furthermore, our evaluation shows that even IM and ETM can fail when challenged with large-scale unfiltered real-life events logs, as shown in the case of the PRT11 log. Among the other APDAs tested, S-HM$_6$ performed well in several accuracy measures, but it was impaired by its inability to systematically discover sound models. Lastly, regarding the complexity of the discovered process models, IM and ETM stood out again, thanks to their guarantee of discovering fully structured process models. However, such hard constraint showed in our benchmark and in previous studies [42] to be a limitation when the process model to discover is not fully structured itself, resulting in the inability to achieve both high fitness and high precision scores and the necessity to trade one of the two for the other.

To conclude, for those approaches we assessed, we could not identify a unique winner, since the best APDAs showed to either maximize fitness or precision. Nevertheless, it can be noted that there has been significant progress in this field in the past five years. Indeed, IM and ETM outperformed the discovery approaches developed in the previous decade, as well as their extensions (i.e., AGNEs Miner and S-HM$_6$), yet leaving space for further improvements. Specifically, the desirable APDA would be the one able to balance fitness and precision achieving high F-scores without compromising the complexity or the soundness of the discovered process model. Designing such an APDA is the major goal of this thesis.

5. SPLIT MINER

In this chapter, [1] we focus on our **RQ2**: *how to strike a trade-off between the various quality dimensions in automated process discovery in an* effective *and* efficient *manner?*

The results we discussed in our SLR and benchmark (respectively Chapter 3 and 4) highlighted the lack of a process discovery approach that efficiently (i.e. in terms of execution time) discovers procedural process models having a balanced fitness and precision and achieving high F-scores, while maintaining the model as simple as possible and guaranteeing soundness.

We address such a gap by proposing an APDA specifically designed to produce simple process models, while balancing fitness and precision. Our proposal combines a novel approach to filter the directly-follows graph (DFG) induced by an event log, with an approach to identify combinations of split gateways that capture the concurrency, conflict and causal relations between neighbouring nodes in the DFG. Given this focus on discovering split gateways, we named our novel process discovery approach *Split Miner*. In its basic variant, Split Miner generates BPMN process models with OR-join gateways. Empirical studies have shown that OR-join gateways induce higher cognitive overload on process models users than the alternative AND-join and XOR-join gateways, which have a simpler semantics. Hence, well-accepted guidelines for process modeling (introduced in Section 4.2) recommend that OR-join gateways should be used sparingly [134]. Accordingly, Split Miner incorporates an algorithm to replace OR-join gateways with AND-join or XOR-join gateways, while guaranteeing that the resulting process model remains sound (in the case of acyclic process models), and deadlock-free in all cases. According to the results of our SLR and benchmark, Split Miner is the first APDA that produces process models that are guaranteed to be deadlock-free, while not restricted to producing block-structured process models only.

The rest of the chapter is structured as follows. Section 5.1 presents Split Miner and its underlying algorithms, while Section 5.2 formally analyses the semantic properties (deadlock-freedom and soundness) of the process models discovered by Split Miner. Section 5.3 discusses the empirical evaluation of Split Miner, which relies on the benchmark framework we presented in Chapter 4. Finally, Section 5.4 summarises the chapter.

5.1. Approach

Starting from a log, Split Miner produces a BPMN model in six steps (cf. Fig. 14). Like the Heuristics Miner and Fodina, the first step is to construct the DFG, but unlike these latter, Split Miner does not immediately filter the DFG. Instead, it analyzes it to detect self-loops and short-loops (which are known to cause problems

---

[1] Corresponding to [17, 18].

in DFG-based process discovery methods) and to discover concurrency relations between pairs of tasks. In a DFG, a concurrency relation between two tasks, e.g. a and b, shows up as two arcs: one from a to b and another from b to a, meaning that causality and concurrency are mixed up. To address this issue, whenever a likely concurrency relation between a and b is discovered, the arcs between these two tasks are pruned from the DFG. The result is called: *pruned DFG* (PDFG). In the third step, a filtering algorithm is applied on the PDFG to strike a balance between fitness and precision maintaining low control-flow complexity. In the fourth step, split gateways are discovered for each task in the filtered PDFG with more than one outgoing arc. Similarly, in the fifth step, join gateways are discovered from tasks with multiple incoming arcs. Lastly, if any OR-joins were discovered, they are removed (whenever possible).

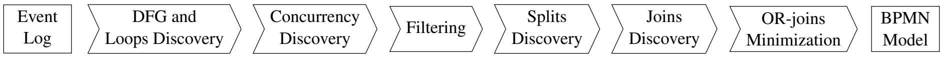


Figure 14: Overview of the proposed approach.

5.1.1. Directly-Follows Graph and Short-Loops Discovery

Split Miner takes as input an event log defined as follows.

Definition 5.1.1 (Event Log)**.** *Given a set of events* Ξ*, an* event log $\mathscr{L}$ *is a multiset of traces as* $\mathscr{T}$*, where a trace* $t \in \mathscr{T}$ *is a sequence of events* $t = \langle \xi_1, \xi_2, \ldots, \xi_n \rangle$*, with* $\xi_i \in \Xi, 1 \leq i \leq n$*. Additionally, each event has a label* $\mathrm{l} \in L$ *and it refers to a task executed within a process, we retrieve the label of an event with the function* $\lambda : \Xi \rightarrow L$*, using the notation* $\xi^l = \lambda(\xi)$*.*

For the remaining, we assume all the traces of an event log have the same start event and the same end event. This is guaranteed by a simple pre-processing of the event log, to be compliant with the third of the 7PMG (Section 4.2), i.e. "*use one start event for each trigger and one end event for each outcome*".

Given the set of labels $L = \{a,b,c,d,e,f,g,h\}$, a possible log is: $\mathscr{L} = \{\langle a,b,c,g,e,h\rangle^{10}, \langle a,b,c,f,g,h\rangle^{10}, \langle a,b,d,g,e,h\rangle^{10}, \langle a,b,d,e,g,h\rangle^{10}, \langle a,b,e,c,g,h\rangle^{10}, \langle a,b,e,d,g,h\rangle^{10}, \langle a,c,b,e,g,h\rangle^{10}, \langle a,c,b,f,g,h\rangle^{10}, \langle a,d,b,e,g,h\rangle^{10}, \langle a,d,b,f,g,h\rangle^{10}\}$; this log contains 10 distinct traces, each of them recorded 10 times.

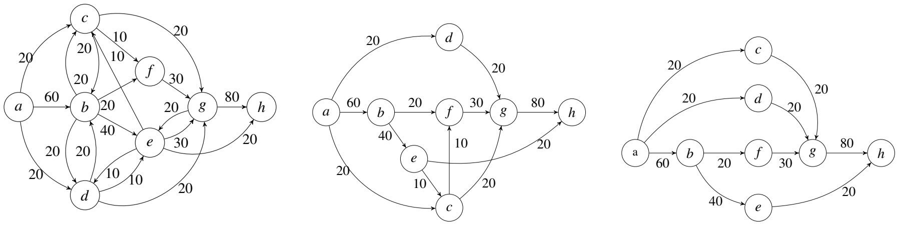
Starting from a log, we construct a DFG in which each arc is annotated with a frequency, based on the following definitions.

Definition 5.1.2 (Directly-Follows Frequency)**.** *Given an event log* $\mathscr{L}$*, and two events labels* $\mathrm{l}_1, \mathrm{l}_2 \in L$*, the directly-follows frequency between* l_1 *and* l_2 *is* $|\mathrm{l}_1 \rightarrow \mathrm{l}_2| = | \{(\xi_i, \xi_j) \in \Xi \times \Xi \mid \xi_i^l = \mathrm{l}_1 \wedge \xi_j^l = \mathrm{l}_2 \wedge \exists t \in \mathscr{L} \mid \exists \xi_x \in t[\xi_x = \xi_i \wedge \xi_{x+1} = \xi_j]]\} |$.

Definition 5.1.3 (Directly-Follows Graph)**.** *Given an event log* $\mathscr{L}$*, its* Directly-Follows Graph (DFG) *is a directed graph* $\mathscr{G} = (N, E)$*, where* N *is the non-empty set of nodes*[2]*, for which a bijective function* $l : N \mapsto L$ *exists, where* n^{l} *retrieve*

---

[2] Each node of the graph represents a task.



(a) Initial state. (b) After pruning. (c) After filtering.

Figure 15: Processing of the directly-follows graph.

the label of n, and E is the set of edges $E = \{(a,b) \in N \times N \mid \left|a^{\mathrm{l}} \rightarrow b^{\mathrm{l}}\right| > 0\}$. *Moreover, given a node* $n \in N$ *we use the operator* $\bullet n = \{(a,b) \in E \mid b = n\}$ *and* $n\bullet = \{(a,b) \in E \mid a = n\}$ *to retrieve (respectively) the set of incoming and outgoing edges.*

Given the DFG, we then detect self-loops and short-loops (i.e. loops involving only one and two tasks resp.) since these are known to cause problems when detecting concurrency [141]. A self-loop exists if a node has an arc towards itself in the DFG: $|a \rightarrow a|$. Short-loops and their frequencies are detected in the log as follows.

Definition 5.1.4 (Short-Loop Frequency)**.** *Given an event log* $\mathscr{L}$*, and two events labels* $\mathrm{l}_1, \mathrm{l}_2 \in L$*, we define the number of times a short-loop pattern occurs* $|\mathrm{l}_1 \leftrightarrow \mathrm{l}_2| = |\{(\xi_i, \xi_j, \xi_k) \in \Xi \times \Xi \times \Xi \mid \xi_i^l = \mathrm{l}_1 \wedge \xi_j^l = \mathrm{l}_2 \wedge \xi_k^l = \mathrm{l}_1 \wedge \exists t \in \mathscr{L} \mid \exists \xi_x \in t[\xi_x = \xi_i \wedge \xi_{x+1} = \xi_j \wedge \xi_{x+2} = \xi_k]\}|$.

In the following, with abuse of notation, given $a, b \in N$ we use $|a \rightarrow b|$ instead of $\left|a^{\mathrm{l}} \rightarrow b^{\mathrm{l}}\right|$ and $|a \leftrightarrow b|$ instead of $\left|a^{\mathrm{l}} \leftrightarrow b^{\mathrm{l}}\right|$.

Given two tasks a and b, a short-loop ($a \circlearrowright b$) exists iff the following conditions hold:

$$|a \rightarrow a| = 0 \quad \wedge \quad |b \rightarrow b| = 0 \tag{5.1}$$

$$|a \leftrightarrow b| + |b \leftrightarrow a| \neq 0 \tag{5.2}$$

Condition 5.1 guarantees that neither a nor b are in a self-loop, otherwise the short-loop evaluation may not be reliable. Indeed, if we consider a model containing a concurrency between a self-loop a and a normal task b, traces recorded during the execution of the process may contain the sub-trace $\langle a, b, a \rangle$ (which also characterize $a \circlearrowright b$). Discarding this latter case fulfilling Condition 5.1, we use Condition 5.2 to ensure $a \circlearrowright b$.

Self-loop are trivially removed from the DFG and restored in the output BPMN model at the end. Fig. 15a shows the DFG built from the example event log $\mathscr{L}$. In this log, there are no self-loops nor short-loops.

5.1.2. Concurrency Discovery

Given a DFG and two tasks a and b, such that neither a nor b is a self-loop[3], we postulate a and b are concurrent ($a \| b$) iff three conditions hold:

$$|a \rightarrow b| > 0 \quad \wedge \quad |b \rightarrow a| > 0 \tag{5.3}$$

$$|a \leftrightarrow b| + |b \leftrightarrow a| = 0 \tag{5.4}$$

$$\frac{||a \rightarrow b| - |b \rightarrow a||}{|a \rightarrow b| + |b \rightarrow a|} < \varepsilon \quad (\varepsilon \in [0,1]) \tag{5.5}$$

Condition 5.3 captures the basic requirement for $a \| b$. Indeed, the existence of edges $e_1 = (a,b)$ and $e_2 = (b,a)$ entails that a and b can occur in any order. However, this is not sufficient to postulate concurrency since this relation may hold in three cases: (i) a and b form a short-loop; (ii) a and b are concurrent; or (iii) e_1 or e_2 occurs highly infrequently and can thus be ignored. Case (i) is avoided by Condition 5.4. Since this latter is the opposite of Condition 5.2, it guarantees $\neg a \circlearrowleft b$. This leaves us with cases (ii) and (iii). We use Condition 5.5 to disambiguate between the two cases: if the condition is true we assume $a \| b$, otherwise we fall into case (iii). The intuition behind Condition 5.5 is that two tasks are concurrent the values of $|a \rightarrow b|$ and $|b \rightarrow a|$ should be as close as possible, i.e. both interleavings are observed with similar frequency. Therefore, the smaller is the value of ε the more balanced have to be the concurrency relations in order to be captured. Reciprocally, setting ε to 1 would catch all the possible concurrency relations.

Whenever we find $a \| b$, we remove e_1 and e_2 from E, since there is no causality but instead there is concurrency. On the other hand, if we find that either e_1 or e_2 represents infrequent behavior we remove the least frequent of the two edges. The output of this step is a *pruned DFG*.

Definition 5.1.5 (Pruned DFG)**.** *Given a* DFG $\mathscr{G} = (N,E)$*, a* Pruned DFG *(PDFG) is a connected graph* $\mathscr{G}_p = (N,E_p)$*, where* E_p *is the set of edges* $E_p = E \setminus \{(a,b) \in E \mid a \| b \vee (\neg a \| b \wedge (b,a) \in E \wedge |a \rightarrow b| < |b \rightarrow a|)\}$.

In the example in Fig. 15a, we can identify four possible cases of concurrency: (b,c), (b,d), (d,e), (e,g). Setting $\varepsilon = 0.2$, we capture the following concurrency relations: $b \| c$, $b \| d$, $d \| e$, $e \| g$. The resulting PDFG is shown in Fig. 15b.

5.1.3. Filtering

In order to derive a sound, simple, and accurate BPMN process model from a PDFG, the latter must satisfy three properties. First, each node of the PDFG must be on a path from the single start node (source) to the single end node (sink). This property is necessary to ensure a sound process model (no deadlocks and no lack of synchronization). Second, for each node, its path from source to sink

---

[3] We favor self-loops over concurrency.

must be the one having maximum capacity. In our context, the capacity of a path is the frequency of the least frequent edge of the path. This property is meant to maximize fitness, since the capacity of a path matches the number of traces that can be replayed on that path. Third, the number of edges of the PDFG must be minimal. This property minimizes CFC and maximizes precision, since the number of edges is proportional to the branching factor (used to calculate the CFC) and to the amount of allowed behavior.

To satisfy these three properties, we designed a variant of Dijkstra's shortest path algorithm [142]. The main differences between Dijkstra's algorithm and ours are the following: (i) during the exploration of the graph we do not propagate the length of the paths, but their capacities; (ii) Dijkstra solves a problem of minimization, whilst we solve a problem of maximization. Additionally, since we want to guarantee that each node is reachable from the source and can reach the sink (i.e. on a path from source to sink), we perform a double breadth-first exploration: forward (source to sink) and backward (sink to source). During the forward exploration, for each node of the PDFG we discover its maximum source-to-node capacity (*forward capacity*), and its incoming edge granting such forward capacity (*best incoming edge*). Similarly, during the backward exploration, we discover maximum node-to-sink capacities (*backward capacities*), and outgoing edges (*best outgoing edge*). Through this algorithm we satisfy the first and second property, and we set a limit to the maximum number of edges retained in our PDFG, which is always less than $2|T|$ (i.e. each node will have at most one incoming and one outgoing edge). However, the limited number of edges may reduce the amount of behaviour that the final model can replay, and consequently its fitness. To strike a trade-off between fitness and precision, we introduce a frequency threshold which let the user balance the two metrics. Precisely, we compute the η percentile over the frequencies of the most frequent incoming and outgoing edges of each node, and we retain those edges with a frequency exceeding the threshold. It is important to notice, that the percentile is not taken over the frequencies of all the edges in E_p since otherwise we would simply retain η percentage of all the edges.

Algorithm 1 shows in detail how we achieve the objectives listed above. Given as input a PDFG $\mathscr{G}_p = (T, E_p)$ and a percentile value η, we detect the source (i) and the sink (o) of $\mathscr{G}_p$. We initialize the forward and backward capacities of each node to 0, except for the source and the sink, which are supposed infinite (line 3 to 13). Simultaneously, for each node we collect the highest frequency among its incoming edges (f_i) and the highest frequency among its outgoing edges (f_o), which are used to estimate the η percentile (line 14).

After the initialization, we perform the breadth-first forward exploration of $\mathscr{G}_p$ starting from the source i. We use a queue (Q) in order to store the nodes to explore, which at the beginning contains only i, and a set (U) for the unexplored nodes containing all the nodes in T (except for i). Finally, the map E_i will store the best incoming edge of each visited node.

Algorithm 1: Generate Filtered PDFG

```
input : PDFG $\mathscr{G}_p = (T, E_p)$, percentile value $\eta$
output: Filtered PDFG $\mathscr{G}_f = (i, o, T, E_f)$

$i \leftarrow$ the source node of $\mathscr{G}_p$
$o \leftarrow$ the sink node of $\mathscr{G}_p$;
Create a map $C_f : T \rightarrow \mathbb{Z}^+$;
Create a map $C_b : T \rightarrow \mathbb{Z}^+$;
Create a set $F \leftarrow \varnothing$;
for $t \in T$ do
    $C_f[t] \leftarrow 0$;
    $C_b[t] \leftarrow 0$;
    $f_i \leftarrow$ the highest frequency of the incoming edges of $t$;
    $f_o \leftarrow$ the highest frequency of the outgoing edges of $t$;
    add $f_i$ and $f_o$ to $F$;
$C_f[i] \leftarrow$ inf;
$C_b[o] \leftarrow$ inf;
$f_{th} \leftarrow$ the $\eta$ percentile on the values in $F$;
Create a map $E_i : T \rightarrow E_p$;
Create a map $E_o : T \rightarrow E_p$;
DiscoverBestIncomingEdges($\mathscr{G}_p$, $i$, $C_f$, $E_i$);
DiscoverBestOutgoingEdges($\mathscr{G}_p$, $o$, $C_b$, $E_o$);
Create a set $E_f \leftarrow \varnothing$;
for $e \in E_p$ do
    if $(\exists t_e \mid E_i[t_e] = e \vee E_o[t_e] = e) \vee (f_e > f_{th})$ then add $e$ to $E_f$ ;
return $\mathscr{G}_f = (i, o, T, E_f)$;
```

Algorithm 2: Discover Best Incoming Edges

```
input : PDFG $\mathscr{G}_p = (T, E_p)$, Source $i$, Forward Capacities Map $C_f$, Best
        Incoming Edges Map $E_i$
Create a queue $Q$;
Create a set $U \leftarrow T \setminus \{i\}$;
Add $i$ to $Q$;
while $Q \neq \varnothing$ do
    $p \leftarrow$ first node in $Q$;
    remove $p$ from $Q$;
    for $e \in p\bullet$ do
        $n \leftarrow$ target of $e$;
        $f_e \leftarrow$ frequency of $e$;
        $C_{max} \leftarrow$ Min($C_f[p]$, $f_e$);
        if $C_{max} > C_f[n]$ then
            $C_f[n] \leftarrow C_{max}$;
            $E_i[n] \leftarrow e$;
            if $n \notin Q \cup U$ then add $n$ to $U$ ;
        if $n \in U$ then
            remove $n$ from $U$;
            add $n$ to $Q$;
```

Algorithm 3: Discover Best Outgoing Edges

```
input : PDFG $\mathscr{G}_p = (T, E_p)$, Sink $o$, Backward Capacities Map $C_b$, Best
        Outgoing Edges Map $E_o$
1  Add $o$ to $Q$;
2  $U \leftarrow T \setminus \{o\}$;
3  while $Q \neq \varnothing$ do
4      $n \leftarrow$ first node in $Q$;
5      remove $n$ from $Q$;
6      for $e \in \bullet n$ do
7          $p \leftarrow$ source of $e$;
8          $f_e \leftarrow$ frequency of $e$;
9          $C_{max} \leftarrow$ Min($C_b[n]$, $f_e$);
10         if $C_{max} > C_b[p]$ then
11             $C_b[p] \leftarrow C_{max}$;
12             $E_o[p] \leftarrow e$;
13             if $p \notin Q \cup U$ then add $p$ to $U$ ;
14         if $p \in U$ then
15             remove $p$ from $U$;
16             add $p$ to $Q$;
```

Using a FIFO[4] policy, we start the exploration of the nodes in Q. When a node p is removed from Q, we analyse its outgoing edges (line 7) and their targets (successors of p). Specifically, for each successor n we evaluate the possible maximum capacity (C_{max}), line 10, that is the minimum between the capacity of its predecessor p ($C_f[p]$) and the frequency of the incoming edge (f_e). Successively, we update the best incoming edge of n ($E_i[n]$) and its current maximum capacity ($C_f[n]$) if C_{max} is greater than $C_f[n]$, line 11.

Since a change of $C_f[n]$ may entail a change of n successors' maximum capacities, we mark n as unexplored if previously explored (n is added to U, line 14). Finally, if n is unexplored, we add it to the tail of Q (line 17). We then perform the backward exploration starting from o (line 1 to 16). The algorithm concludes retaining the best incoming and outgoing edges, and those with a frequency above the threshold f_{th} (line 21).

Though the third property (i.e. $\mid E_f \mid < 2 \mid T \mid$) can only be guaranteed for $\eta = 1$, the usage of η is meant to balance fitness and precision. Since, the lower is the value of η the more edges may be retained, resulting in a higher fitness at the cost of lower precision and higher control-flow complexity.

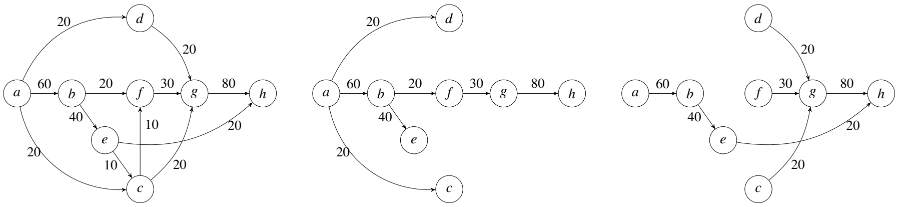
Figure 15c shows the output of the filtering algorithm when applied to the PDFG previously obtained (Figure 15b). The results of the forward and backward

---

[4] First-in first-out.

Node	C_f	E_i	C_b	E_o
a	∞	-	20	(a,b)
b	60	(a,b)	20	(b,e)
c	20	(a,c)	20	(c,g)
d	20	(a,d)	20	(d,g)
e	40	(b,e)	20	(e,h)
f	20	(b,f)	30	(f,g)
g	20	(f,g)	80	(g,h)
h	20	(g,h)	∞	-

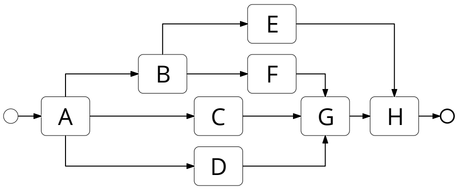
Table 15: Filtering algorithm example.



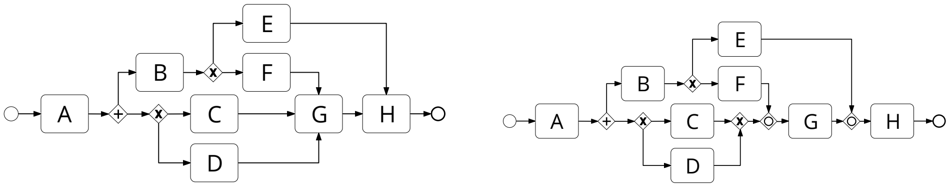
(a) PDFG. (b) Selection of Algorithm 2. (c) Selection of Algorithm 3.

Figure 16: Best incoming and outgoing edges selected by Algorithm 2 and 3 from the PDFG.

explorations (mappings C_f, E_i, and C_b, E_o) are shown in Table 15. As consequence of retaining the best incoming and outgoing edges for each node, the filtering algorithm would drop the edges: (e,c) and (c,f). Regardless of the value assigned to η, only these two edges would be removed. This is due to the design of our filtering algorithm, which cannot remove any incoming (outgoing) edge that is the only incoming (outgoing) edge of a node. In our working example, Algorithm 2 makes a selection only for the incoming edges of the nodes: c (from a and e); f (from b and c); g (from c, d, and f); h (from e and g). While, Algorithm 3 makes a decision only for the outgoing edges of the nodes: a (to b, c, and d); b (to f and e); c (to f and g); e (to c and h). During the forward breadth-first exploration, Algorithm 2 selects the edges (a,c); (b,f); (f,g); (g,h); ensuring the forward capacity of c, f, g, and h are maximal (20, 20, 20, 20). If we would keep only the edges selected by Algorithm 2, the filtered PDFG would be the one in Figure 16b, where some nodes (c, d, and e) are not on a path from source to sink. This is the reason why a backward breadth-first exploration is required, and implemented via Algorithm 3, which selects the additional edges (see Figure 16c) required to guarantee that each node is on a path from source to sink, with path capacity (i.e. the minimum of forward and backward capacities) maximal. The union of the edges selected by Algorithm 2 and 3 is the final output, as shown in Figure 15b.



(a) Initial state.



(b) After splits discovery. (c) After joins discovery.

Figure 17: Processing of the BPMN model.

5.1.4. Filtered PDFG to BPMN Process Model

Once the processing of the DFG is completed, we can start the conversion from the filtered PDFG to the BPMN process model.

Definition 5.1.6 (BPMN process model)**.** *A* BPMN process model *(or BPMN model) is a connected graph* $\mathscr{M} = (i, o, T, G, E_m)$*, where i is the start event, o is the end event, T is a non-empty set of tasks,* $G = G^+ \cup G^\times \cup G^\circ$ *is the union of the set of AND gateways (*G^+*), the set of XOR gateways (*$G^\times$*) and the set of OR gateways (*G°*), and* $E_m \subseteq (T \cup G \cup \{i\}) \times (T \cup G \cup \{o\})$ *is the set of edges. Further, given* $g \in G$*, g is a* split *gateway if it has more than one outgoing edge, i.e.* $\mid g\bullet \mid > 1$*, or a* join *gateway if it has more than one incoming edge, i.e.* $\mid \bullet g \mid > 1$.

Algorithm 4 highlights the main parts of the conversion. Specifically, we create a start and an end event (lines 1 and 2). Then, we initialize the set of tasks and edges, respectively as the set of nodes of the filtered PDFG, and as the set of the edges of the filtered PDFG plus two new edges: one connecting the start event with the former source of the DFG, and one connecting the former sink of the DFG to the end event (line 6). Lastly, an empty set of gateways is created, which will be filled through the following three steps: split discovery, join discovery and ORs replacement. Figure 17a shows the initialization of the BPMN model obtained from the filtered PDFG of Figure 15c.

5.1.5. Splits Discovery

To generate the split gateways, we rely on the concurrency relations identified during the second step of our approach (section 5.1.2). The splits discovery is based on the idea that tasks directly following (successors of) the same split gateway are concurrent to the same set of tasks which do not directly follow such gateway. With a reference to Figure 18b, given that tasks c and d are successors of the gateway and_1, they are both concurrent to tasks e, f, g, due to gateway and_3 (i.e. $c \| e$,

Algorithm 4: Filtered PDFG to BPMN process model

```
input  : Filtered PDFG $\mathscr{G}_f = (N, E_f)$
output: BPMN process model $\mathscr{M} = (i, o, G, T, E_m)$
Create a start event $i$;
Create an end event $o$;
Create a set $T \leftarrow N$;
$s \leftarrow t \in T \mid |\bullet t| = 0$;
$e \leftarrow t \in T \mid |t\bullet| = 0$;
Create a set $E_m \leftarrow E_f \cup \{(i, s), (e, o)\}$;
Create a set $G \leftarrow \varnothing$;
Create a BPMN process model $\mathscr{M} \leftarrow (i, o, T, G, E_m)$;
DiscoverSplits($\mathscr{M}$);
DiscoverJoins($\mathscr{M}$);
ReplaceORs($\mathscr{M}$);
return $\mathscr{M}$;
```

Algorithm 5: Discover Splits

```
input  : BPMN process model $\mathscr{M} = (i, o, T, G, E_m)$
for $t \in T$ do
    if $| t\bullet | > 1$ then
        Create a set $S \leftarrow$ d-successors of $t$;
        Create a map $C : S \rightarrow 2^S$;
        Create a map $F : S \rightarrow 2^S$;
        for $s_1 \in S$ do
            $C[s_1] \leftarrow \{s_1\}$;
            $F[s_1] \leftarrow \varnothing$;
            for $s_2 \in S$ do
                if $(s_2 \neq s_1 \wedge s_2 \| s_1)$ then add $s_2$ to $F[s_1]$;
                ;
        remove from $E_m$ the outgoing edges of $t$ ;
        while $|S| > 1$ do
            discoverXORsplits($\mathscr{M}, S, C, F$);
            discoverANDsplits($\mathscr{M}, S, C, F$);
        $s \leftarrow$ unique element of $S$;
        add an edge from $t$ to $s$;
```

Algorithm 6: Discover XOR-splits

```
input : BPMN $\mathscr{M}$, Set $S$, Map $C$, Map $F$
do
    Create a set $X \leftarrow \varnothing$;
    for $s_1 \in S$ do
        Create a set $C_u \leftarrow C[s_1]$;
        for $s_2 \in S$ do
            if $F[s_1] = F[s_2] \wedge s_1 \neq s_2$ then
                add $s_2$ to $X$;
                $C_u \leftarrow C_u \cup C[s_2]$;
        if $X \neq \varnothing$ then
            add $s_1$ to $X$;
            break;
    if $X \neq \varnothing$ then
        Create an XOR gateway $g_\times$;
        add $g_\times$ to $G^\times$;
        for $s \in X$ do
            add an edge from $g_\times$ to $s$;
            remove $s$ from $S$;
        add $g_\times$ to $S$;
        $F[g_\times] \leftarrow F[s_1]$;
        $C[g_\times] \leftarrow C_u$;
while $X \neq \varnothing$;
```

Algorithm 7: Discover AND-splits

```
input : BPMN $\mathscr{M}$, Set $S$, Map $C$, Map $F$
do
    Create a set $A \leftarrow \varnothing$;
    for $s_1 \in S$ do
        Create a set $C_u \leftarrow C[s_1]$;
        Create a set $F_i \leftarrow F[s_1]$;
        Create a set $CF_{s_1} \leftarrow C[s_1] \cup F[s_1]$;
        for $s_2 \in S$ do
            Create a set $CF_{s_2} \leftarrow C[s_2] \cup F[s_2]$;
            if $CF_{s_1} = CF_{s_2} \wedge s_1 \neq s_2$ then
                add $s_2$ to $A$;
                $C_u \leftarrow C_u \cup C[s_2]$;
                $F_i \leftarrow F_i \cap F[s_2]$;
        if $A \neq \varnothing$ then
            add $s_1$ to $A$;
            break;
    if $A \neq \varnothing$ then
        Create an AND gateway $g_+$;
        add $g_+$ to $G^+$;
        for $s \in A$ do
            add an edge from $g_+$ to $s$;
            remove $s$ from $S$;
        add $g_+$ to $S$;
        $C[g_+] \leftarrow C_u$;
        $F[g_+] \leftarrow F_i$;
while $A \neq \varnothing$;
```

$c||f$, $c||g$, and $d||e$, $d||f$, $d||g$). Similarly, since tasks a and b are successors of the gateway xor_1, they share the same empty set of concurrent relations, because a, b and all the other tasks are mutually exclusive due to gateway xor_3. Knowing which tasks are successors of the same gateway, we can identify the gateway's type by checking whether its successors are concurrent or mutually exclusive, e.g. c and d are concurrent (i.e. $c||d$), whilst a and b are mutually exclusive (i.e. $\neg\, a||b$).

Before explaining in details how we discover a hierarchy of splits, we need to define the following concepts.

Definition 5.1.7 (Split-task)**.** *Given a BPMN model* $\mathscr{M} = (i, o, T, G, E_m)$*, a* split-task *is a task* $t \in T$*, such that* $|t\bullet| > 1$.

Definition 5.1.8 (P-successor of a Split-task)**.** *Given a BPMN model* $\mathscr{M} = (i, o, T, G, E_m)$ *and a split-task* $t \in T$*, a* p-successor *of* t *is a task or gateway* $s \in T \cup G$ *such that there exists a path from* t *to* s*. Further, if* $s \in T$ *we say* s *is a* t-successor*, whilst if* $s \in G$*,* s *is a* g-successor.

Definition 5.1.9 (D-successor of a Split-task)**.** *Given a BPMN model* $\mathscr{M} = (i, o, T, G, E_m)$ *and a split-task* $t \in T$*, a* d-successor *of* t *is a task* $s \in T \cup G$ *such that there exists an edge from* t *to* s*. A d-successor is always a p-successor, but not vice-versa.*

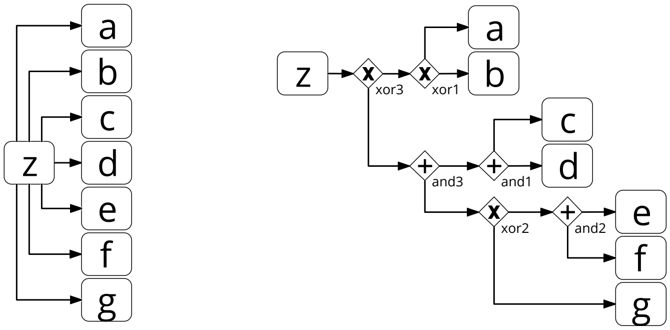
Definition 5.1.10 (Successor Cover)**.** *Given a BPMN model* $\mathscr{M} = (i, o, T, G, E_m)$*, a split-task* $t \in T$ *and a p-successor* s *of* t*, the* cover *of* s *is the subset* C_s *of the t-successors of* t *such that for each* $c \in C_s$ *there exists a path from* t *to* c *that visits* s*. The following properties are always true:* $s \in C_s$ *and* $C_s \cap G = \varnothing$.

Intuitively, given a split-task and a p-successor, the cover of the p-successor is a set containing all the tasks that can be visited on the paths from the split-task to the p-successor (including the p-successor itself).

Definition 5.1.11 (Successor Future)**.** *Given a BPMN model* $\mathscr{M} = (i, o, T, G, E_m)$*, a split-task* $t \in T$ *and a p-successor* s*, the* future *of* s *is the subset* F_s *of the t-successors of* t *such that* $f \in F$ *iff* $f||c, \forall c \in C_s$.

Intuitively, given a split-task and a p-successor s, the future of the p-successor is the set containing all the other p-successors of the split-task that can be executed in the same process instance of s.

To describe how Algorithm 5 discovers a hierarchy of splits we use the example in Figure 18, assuming the concurrency relations showed in Figure 18b. Given as input a BPMN model to Algorithm 5, we look for split-tasks (line 1), e.g. z (Figure 18a). We retrieve the d-successors of z (line 3), and for each d-successor, e.g. e, we detect its *cover* and its *future* (maps $C[s]$ and $F[s]$, lines 4 and 5). In our example, the cover and the future of e are the sets $\{e\}$ and $\{c, d, f\}$, respectively (since we assumed $e||c$, $e||d$ and $e||f$). Table 16 shows how the sets $C[s]$ and $F[s]$ evolves during the execution of the algorithm. After computing *cover* and *future* of each d-successor (line 6 to 10), we use them to discover XOR-splits and AND-splits in two phases (lines 14 and 15). In the first phase – Algorithm 6 – we look for all the d-successors sharing the same *future* (line 6). Whenever we find any (line 9), we introduce an XOR-split preceding these p-successors, which replaces



(a) Before (b) After

Figure 18: Splits discovery example.

them as d-successor of z. This gateway has as *future* the same shared *future* of the selected d-successors, and as *cover* the union of their *covers*. This is in-line with our initial idea, since d-successors having the same *future* are successors of the same gateway and are not concurrent. Otherwise, if they were concurrent, they would be in each other *futures* and their *futures* would differ.[5] We repeat this operation until no further XOR-splits are identified (line 21). Once all possible XOR-splits are discovered, we move toward the second phase, i.e. the discovery of AND-splits – Algorithm 7.

Unlike the XOR-splits, to identify AND-splits we cannot rely only on the d-successors' *futures*. Instead, we select the d-successors having the same set of nodes resulting from the union of their *cover* and *future* (line 9 to 15). Then, an AND-split is introduced before these d-successors. This AND-split has as *future* the intersection of the *futures* of the set of the selected d-successors and as *cover* the union of their *covers*. Likewise for the XOR-split, the AND-split becomes a new d-successor replacing the p-successors that will now belong to its cover. The discovery of the AND-splits is still in-line with our initial idea. Indeed, given a set of candidate AND-split successors (i.e. d-successors having a common subset of their *futures*), and removing the common subset from their *futures*, the genuine AND-split successors will be those for which their *cover* is contained in the *future* of each other candidate successor of the AND-split (i.e. successors of the same AND-split must be concurrent each others). In our example (Table 16), c and d have as shared future the subset $F_s = \{e, f, g\}$. If we remove this subset, we would see that the cover of c matches the future of d and vice-versa (i.e. $c \| d$ and $d \| c$). The fact that we look for d-successors having the same union of their *cover* and *future* is a mere computational optimization, since by construction the intersection of the covers of two different d-successors is always empty, and the intersection between the cover and the future of a d-successor is always empty.

We repeat Algorithms 6 and 7 until the split-task becomes a normal task, hav-

---

[5] This happens because by construction a task cannot belong to its own future.

D-successor	C	F
a	a	-
b	b	-
c	c	d, e, f, g
d	d	c, e, f, g
e	e	f, c, d
f	f	e, c, d
g	g	c, d
xor_1	a, b	-
and_1	c, d	e, f, g
e	e	f, c, d
f	f	e, c, d
g	g	c, d
xor_1	a, b	-
and_1	c, d	e, f, g
and_2	e, f	c, d
g	g	c, d
xor_1	a, b	-
and_1	c, d	e, f, g
xor_2	e, f, g	c, d
xor_1	a, b	-
and_3	c, d, e, f, g	-
xor_3	a, b, c, d, e, f, g	-

Table 16: Splits discovery example.

ing just one d-successor (Algorithm 5, line 13). Figure 17b shows the output of this step for our working example, when we give as input to Algorithm 5 the BPMN models showed in Figure 17a.

5.1.6. Joins Discovery

Once all the split gateways have been placed, we can discover the join gateways. To do so, we rely on the Refined Process Structure Tree (RPST) [143] of the current BPMN model. The RPST of a process model is a tree where its nodes represent the single-entry single-exit (SESE) fragments of the process model and its edges denote a containment relation between SESE fragments. Specifically, the children of a SESE fragment are its directly contained SESE fragments, whilst SESE fragments on different branches of the tree are disjoint. Since each SESE fragment is a subgraph of the process model, and the partition of the process model into SESE fragments is made in terms of edges, a single node (of the process model) can be shared by multiple SESE fragments. Further, each SESE fragment can be of one of the four types: a *trivial* fragment consists of a single edge; a *polygon* is a sequence of fragments; a *bond* is a fragment where all the children fragments share two common nodes, one being the entry and the other being the exit of the bond; any other fragment is a *rigid*. Lastly, each SESE can be classified as *homogeneous* if the gateways it contains (and are not contained in any of its SESE children) are all of the same type (e.g. only XOR-gateways), or *heterogeneous* if such gateways have different types.

To explain Algorithm 8, we need to introduce the concept of *loop-join*.

Definition 5.1.12 (Loop-edge and Loop-joins)**.** *Given a BPMN model* $\mathscr{M} = (i, o, T, G, E_m)$*, and an edge* $e = (a, b) \in E_m$*, e is a* loop-edge *iff a is a node topologically deeper than b and there exists a path from b to a. Further, if* $|\bullet b| > 1$*, we refer to b as* loop-join.

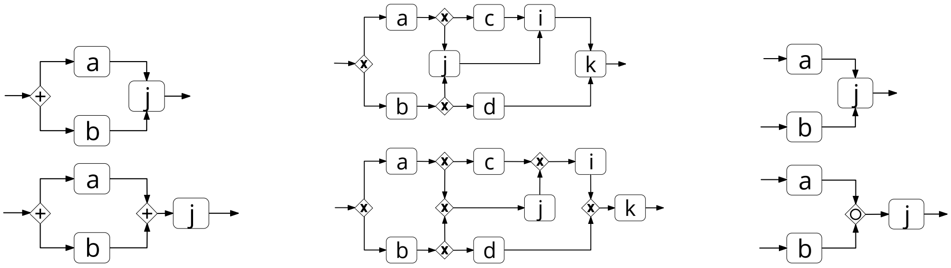
The first step of Algorithm 8 is to generate the RPST of the input BPMN model. Then, we add all the RPST nodes to a queue (Q) ordering the nodes bottom-up, i.e. leaves to root (line 2), and we analyse each of these nodes (which are the SESE fragments composing the BPMN model). Precisely, for each task (t) having multiple incoming edges within the SESE fragment, we create a new join gateway (g) and we redirect all the incoming edges of t to g, line 12. Finally, we set the type of g according to the following rules: if g is a loop-join, it is turned into a XOR; else if t is within a homogeneous SESE fragment we match the type of the homogeneous SESE fragment, otherwise the type is set to OR. These rules guarantee soundness for acyclic models and deadlock-freedom for cyclic models as discussed below.

Figure 19 shows how our approach works for bonds (Fig. 19a), for homogeneous rigids (Fig. 19b), and for all other cases, i.e. heterogeneous rigid (Fig. 19c).

Considering the working example in Fig. 17b, we detect three joins. The first one is the XOR-join at the exit of the bond containing tasks c, d and g. Given that

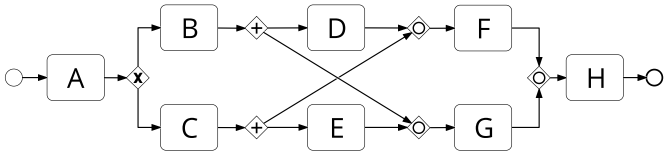
Algorithm 8: Discover Joins

```
input : BPMN process model $\mathscr{M} = (i, o, T, G, E_m)$
generate the RPST from $\mathscr{M}$;
Create a queue $Q \leftarrow$ nodes of the RPST bottom-up ordered;
while $Q \neq \varnothing$ do
    $n \leftarrow$ first node in $Q$;
    remove $n$ from $Q$;
    Create a set $T_n \leftarrow$ tasks composing $n$;
    Create a set $E_n \leftarrow$ edges composing $n$;
    for $t \in T_n$ do
        if $\mid \bullet t \cap E_n \mid > 1$ then
            create a gateway $g$;
            add an edge from $g$ to $t$;
            for $e \in \bullet t \cap E_n$ do  set target of $e$ to $g$;
            ;
            if $\bullet g$ contains backedges then
                set the type of $g$ to XOR;
            else
                if $n$ is homogeneous then
                    set the type of $g$ equal to the type of $n$;
                else
                    set the type of $g$ to OR;
```

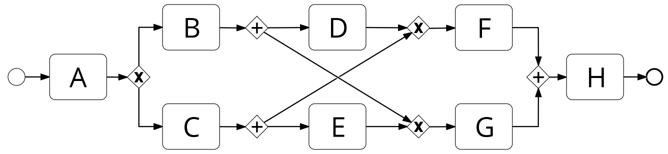


(a) Bond. (b) Homogeneous Rigid. (c) Generic case.

Figure 19: Joins discovery examples.



(a) Before.



(b) After.

Figure 20: OR-joins minimization example.

the entry of this bond is an XOR-split, the bond is XOR-homogeneous, so that the type of the joins is set to XOR. The remaining two joins are within the parent SESE fragment of the bond, which is a heterogeneous rigid, hence, we use two OR-joins. The resulting model is shown in Fig. 17c.

5.1.7. OR-joins Minimization

The approach described in Section 5.1.6 avoids the placement of trivial OR-joins within bonds and homogeneous rigids, but it does not prevent an abuse of OR-joins in case of heterogeneous rigids. Since we aim to be compliant with the 7PMG [134] (Section 4.2), according to the fifth guideline, we should avoid the use of OR gateways where possible. To achieve this goal, we designed an algorithm able to minimize the use of the OR-joins opportunely replacing the trivial ones[6] with XOR or AND joins. Since the algorithm is centred on the concept of *minimal dominator*, we introduce it formally.

Definition 5.1.13 (Minimal Dominator)**.** *Given a BPMN model* $\mathscr{M} = (i, o, T, G, E_m)$*, an OR-join* $j_\circ \in G^\circ$*, and a split gateway* $d \in G$*,* d *is a* dominator *of* $j_\circ$ *iff all the paths from* i *to* $j_\circ$ *visit* d*. Furthermore,* d *is the* minimal dominator *of* $j_\circ$ *iff none of the paths from* d *to* $j_\circ$ *visits other dominators of* $j_\circ$*.*

The procedure for the *OR-joins minimization* detects the trivial OR-joins of the input BPMN model, and replaces them with XOR or AND joins according to their semantics (proof in Section 5.2). Figure 20 shows an example of the OR-joins minimization. We describe the idea behind this procedure before presenting the algorithm.

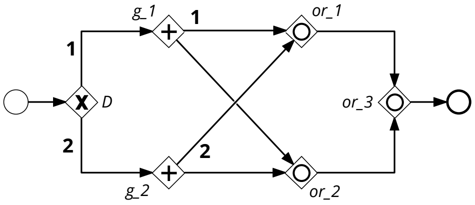
Given a BPMN model $\mathscr{M} = (i, o, T, G, E_m)$, an OR-join $j_\circ$, and its minimal dominator $d \in G$, we know that all tokens that may arrive to one or more incoming edge of $j_\circ$ (*incoming tokens*) are generated by d.

---

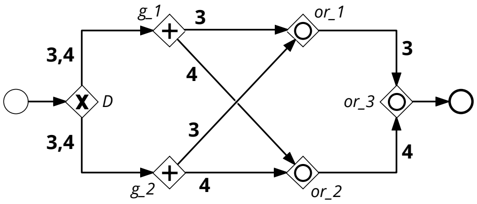
[6] An OR-join is said trivial when its semantic is equivalent to the semantic of an XOR or AND join.

Algorithm 9: Check OR-join Semantic

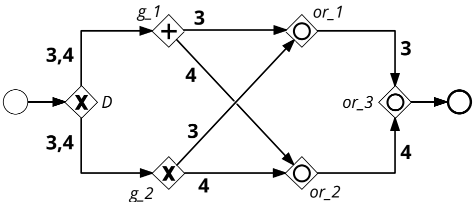
```
input : A BPMN model $\mathscr{M} = (i, o, T, G, E_m)$, Gateway $j_\circ \in G^\circ$
output: Semantic of $j_\circ$
$d \leftarrow$ minimal dominator of $j_\circ$;
Create a set $G_s \leftarrow$ split gateways in any path from $d$ to $j_\circ$;
Create a set $E_s \leftarrow$ outgoing edges of all the gateways in $G_s$;
Create a map $T : E_s \rightarrow 2^{\bullet j_\circ}$;
semantic $\leftarrow$ null;
for $e \equiv (g_s, x) \in E_s$ do
    $T[e] \leftarrow \{e_{ij} \in \bullet j_\circ \mid \exists$ a path from $x$ to $e_{ij}\}$;
    if $T[e] = \varnothing \wedge$ semanticOf($g_s$) = XOR then semantic = XOR;
    ;
for $g \in G_s$ do
    for $e_1 \in g\bullet$ do
        for $e_2 \in g\bullet$ do
            $I \leftarrow T[e_1] \cap T[e_2]$;
            $S_1 \leftarrow T[e_1] \setminus I$;
            $S_2 \leftarrow T[e_2] \setminus I$;
            if $S_1 \neq \varnothing \wedge S_2 \neq \varnothing$ then
                if semantic $! =$ null $\wedge$ semantic $! =$ semanticOf(g) then
                    return OR;
                else
                    semantic $\leftarrow$ semanticOf($g$);
return semantic;
```



(a) Replacing of OR_1 and OR_2.



(b) Replacing of OR_3.



(c) Non-trivial OR_3.

Figure 21: OR-joins minimization algorithm.

By checking the semantic of all the split gateways visited by the incoming tokens in all the paths from d to $j_\circ$, we identify how the incoming tokens split (*split relations*), i.e. mutually exclusive or concurrently. Finally, if the split relations between the incoming tokens of $j_\circ$ are all the same, $j_\circ$ is a trivial OR-join, and it can be replaced with an XOR-join if the split relations are of mutually exclusive type or an AND-join if concurrent. Otherwise, the OR-join is not trivial, and its replacement must be handled differently. Algorithm 9 shows in details how we check the semantic of an OR-join, to decide if we can replace it. Given a BPMN model $\mathscr{M} = (i, o, T, G, E_m)$ and an OR-join $j_\circ$, we retrieve the minimal dominator of $j_\circ$ (d, line 1). We create a set (G_s) containing all the split gateways visited on the paths from d to $j_\circ$; and a set (E_s) containing all the outgoing edges of the split gateways in G_s. Then, for each edge $e \equiv (g_s, x) \in E_s$, we initialize a set ($T[e]$) containing the edges $e_{ij} \in \bullet j_\circ$ such that there exists a path from x to e_{ij} (i.e. $T[e]$ contains the incoming edges of $j_\circ$ that may receive a token from e). During this initialization, we may find an empty set $T[e]$. In such case, if g_s (source of e) is an XOR-split, the empty set $T[e]$ indicates that an incoming token of $j_\circ$ may escape on g_s from e. Therefore, since there exists a case where $j_\circ$ does not receive a token on one of its incoming edges, $j_\circ$ cannot be a trivial AND-join. We record this occurrence setting the possible trivial semantic of the OR-join as exclusive (XOR, line 9). Initialized the sets $T[e]$, we can check the split relations between the incoming tokens of $j_\circ$. Precisely, for each split gateway $g \in G_s$, and for each pair of its outgoing edges (e_1 and e_2, lines 10 to 12), we compute the intersection of $T[e_1]$ and $T[e_2]$. This intersection (I) contains the edges of $\bullet j_\circ$ that may receive incoming tokens either from e_1 or e_2 (i.e. incoming tokens for the edges in I do not split between e_1 and e_2). By removing I from $T[e_1]$ and $T[e_2]$ (sets S_1 and S_2, lines 14 and 15), we detect the incoming tokens that split between e_1 and e_2, i.e. S_1 (S_2) contains the incoming edges of $j_\circ$ that cannot receive tokens from e_2 (e_1). If S_1 and S_2 are both not empty, we identified a split relation between the incoming tokens of the edges in S_1 and the incoming tokens of the edges in S_2. Consequently, the incoming tokens that split between e_1 and e_2 may arrive to different incoming edges of $j_\circ$ with the same semantic of g_s (i.e. the gateway where they split). Therefore, $j_\circ$ is trivial only if all the incoming tokens match that semantic (line 17). Figure 21, shows graphically how Algorithm 9 works on each OR-join of the model in Figure 20a. Specifically, Figure 21a highlights where the incoming tokens of the OR-join OR_1 split. The minimal dominator of OR_1 (as well for OR_2 and OR_3) is the XOR-gateway D. By running Algorithm 9 on OR_1, we detect that the two incoming tokens (namely 1 and 2) split only on D. Since the semantic of D is exclusive, we can replace OR_1 with an XOR-join. The semantic check for OR_2 is equivalent to the one of OR_1. For OR_3 (Figure 21b), its incoming tokens (namely 3 and 4) may split on g_1 and g_2, but they do not split on D. Since g_1 and g_2 are both AND-splits, we can replace OR_3 with and AND-split. In such example, all three OR-joins were trivial. Differently, if g_2 is an XOR-split (figure 21c), OR_1 and OR_2 would still be replaced with XOR-joins, whilst OR_3

would remain an OR-join. This would happen because the incoming tokens of OR_3 split on g_2 (now XOR) and on g_1 (still AND), meaning that their semantics may be either exclusive or concurrent, according to where they would split during the execution of the process model (either on g_1 or g_2).

5.1.8. Time Complexity

Let n be the number of events in the log and m be the number of tasks (distinct nodes of the DFG). The DFG construction is in $O(n)$, since we sequentially read each event and generate the respective node in the graph, simultaneously incrementing the directly-follows and short-loop frequencies. The self-loops discovery is linear on the number of nodes of the DFG, hence in $O(m)$. The short-loops discovery is done on pairs of tasks, so this step is performed in $O(m^2)$. The filtering complexity is dominated by the forward (backward) exploration. It explores each node a number of times equal at most to the number of edges of the graph, and for each node exploration it loops on the outgoing (incoming) edges. In the worst scenario, the maximum number of edges is equal to m^2 (e.g. an edge for each pair of nodes), consequently, the filtering is in $O(m^4)$. The split discovery is in $O(m^4)$, because we may run Algorithm 5 for each node, which executes m times Algorithm 6 and 7, and these latter have two nested loops on m. The join discovery complexity is dominated by the three nested loops: the one on the number of nodes of the RPST, the one on the number of tasks, and the one on the number of edges. Since the RPST contains a number of nodes equal at most to the number of edges of the model, the join discovery is in $O(m^5)$. For the OR-minimization, we run Algorithm 9 for each OR-join – m times, i.e. one join for each task. The complexity of Algorithm 9 is dominated by its three nested loops. The outer loop is on the number of split gateways – bound by m, i.e. one split for each task –, whilst the two inner loops are on the number of edges. Therefore, the OR-minimization is in $O(m^6)$. We can conclude that Split Miner is in $O(n+m^6)$.

5.2. Semantic Properties of the Discovered Model

In this section, we provide formal proofs of some semantic properties of the BPMN process model discovered by Split Miner. Precisely, we show that in the case of *acyclic* BPMN process models, Split Miner guarantees soundness and the absence of any trivial OR-joins. Moreover, for *cyclic* BPMN process models, it is not possible to guarantee the soundness, but only deadlock-freedom. This latter result is ensured by the semantic of the OR-joins [144].

In the following, we refer to the BPMN process model as workflow graph.

Definition 5.2.1 (Workflow graphs). *A* workflow graph *is a triple* $G := (V, E, l)$*, where* (V, E) *is a finite directed graph consisting of a set* V *of nodes and a set* $E \subseteq V \times V$ *of edges, and* $l : V \to \{AND, XOR, OR\}$ *is a partial mapping such that:*

1. *there is exactly one* source *node, where a node* $v \in V$ *is a* source *if and only*

if it has exactly one outgoing edge and no incoming edges,

2. *there is at least one* sink *node, where a node $v \in V$ is a* sink *if and only if it has exactly one incoming edge and no outgoing edges,*
3. *If $l(v)$, $v \in V$, is defined, then v is neither the source nor a sink,*
4. *If $v \in V$ is a gateway, then $l(v)$ is defined, and*
5. *every node $v \in V$ is on a directed path from the source to some sink.*

We chose this formalism to ease the readability of the proofs and their symbolism. It is important to notice that by definition a workflow graph $G := (V, E, l)$ is equivalent to a BPMN process model $\mathscr{M} = (i, o, T, G, E_m)$. Where $\{i\} \cup \{o\} \cup T \cup G \equiv V$, $E_m \equiv E$, i and o are the only source and sink of the workflow graph, and the type of the gateways is identified by the function l.

5.2.1. Preliminaries

Let $G := (V, E, l)$ be a workflow graph. A *state* of G is a mapping $s : E \rightarrow \mathbb{N}_0$.[7]

Definition 5.2.2 (Semantics of workflow graphs)**.** *A* state transition *of a workflow graph $G := (V, E, l)$ is a triple (s_1, v, s_2), also written as $s_1[v\rangle s_2$, where s_1 and s_2 are states of G, $v \in V$, and one of these three conditions holds:*

1. *$l(v) \notin \{XOR, OR\}$ and*

$$s_2(e) = \begin{cases} s_1(e) + 1 & e \in E \text{ is an outgoing edge of } v \\ s_1(e) - 1 & e \in E \text{ is an incoming edge of } v \\ s_1(e) & \text{otherwise.} \end{cases}$$

2. *$l(v) = XOR$ and there exists an incoming edge $e_1 \in E$ of v and an outgoing edge $e_2 \in E$ of v such that:*

$$s_2(e) = \begin{cases} s_1(e) + 1 & e \in E \text{ and } e = e_2 \\ s_1(e) - 1 & e \in E \text{ and } e = e_1 \\ s_1(e) & \text{otherwise.} \end{cases}$$

3. *$l(v) = OR$ for each edge $e' \in E$ and each incoming edge $e \in E$ of v such that $s_1(e') \geq 1$ and $s_1(e) = 0$ there is no directed path from e' to e, and there exists a nonempty set F of outgoing edges of v such that:*

$$s_2(e) = \begin{cases} s_1(e) + 1 & e \in F \\ s_1(e) - 1 & e \in E \text{ is an incoming edge of } v \text{ such that } s_1(e) \geq 1 \\ s_1(e) & \text{otherwise.} \end{cases}$$

Let e be the only outgoing edge of the source of a workflow graph $G := (V, E, l)$. Then, state s of G for which it holds that $s(e) = 1$ and for every $e' \in E$ such that $e' \neq e$ it holds that $s(e') = 0$ is the *initial state* of G. Let s_0 be the initial state of G. A state s is a *reachable state* of G if and only if $s = s_0$ or there is a sequence of nodes $\sigma := \langle v_1, \ldots, v_n \rangle \in V^*$, $n \in \mathbb{N}$, such that for every position i of σ it holds

---

[7] By $\mathbb{N}_0$, we denote the set of all natural numbers including zero.

that $s_{i-1}[v_i\rangle s_i$ is a state transition of G, and $s = s_n$. A state s is a *final state* of G if and only if for every $v \in V$ there exists no state s' of G such that (s, v, s') is a state transition of G. A final state s of G is a *deadlock* if and only if there is an edge $e \in E$ such that $s(e) > 0$ and e is not an incoming edge of some sink of G. A state s of G is *safe* if and only if for all $e \in E$ it holds that $s(e) \leq 1$; otherwise s is *unsafe*. A workflow graph $G := (V, E, l)$ is *safe* if and only if all its reachable states are safe; otherwise G is *unsafe*.

Definition 5.2.3 (Sound worfklow graphs)**.** *A workflow graph G is* sound *if and only if G is safe and has no reachable deadlocks.*

Given a node $v \in V$ of a workflow graph $G := (V, E, l)$, by $\bullet v$ we denote the set of all incoming edges of v. Whereas by $v\bullet$, we denote the set of all outgoing edges of v. In what follows, without loss of generality, we assume that for every vertex v of every workflow graphs it does not hold that $|\bullet v| > 1$ and $|v\bullet| > 1$. A node $v \in V$ is a *gateway* if and only if it holds that $|\bullet v| > 1$ or $|v\bullet| > 1$. A gateway $v \in V$ is a *split* if and only if $|\bullet v| > 1$. A gateway $v \in V$ is a *join* if and only if $|v\bullet| > 1$. Hence, every gateway is either a split or a join, but not both.

Let $G := (V, E, l)$ be a triple, where (V, E) is a finite directed graph consisting of a set V of nodes and a set $E \subseteq V \times V$ of edges, and $l : V \rightarrow \{AND, XOR, OR\}$ is a partial mapping. A *prefix* of G is a triple $G' := (V', E', l')$, denoted by $G' \sqsubseteq G$, where $V' \subseteq V$ is such that if $v \in V'$ then for every $v' \in V$ for which $(v', v) \in E$ it holds that $v' \in V'$, $E' \subseteq E$ is such that (V', E') is a connected graph, and $l' := l|_{dom(l) \cap V'}$. We write $G' \sqsubset G$ if and only if $G' \sqsubseteq G$ and $G' \neq G$.

Let $G := (V, E, l)$ be a prefix of a workflow graph and let $X \subseteq V$ be all the nodes of G such that for every $x \in X$ it holds that $x\bullet = \emptyset$. A *completion* of G is a triple $G' := (V', E', l)$, where $V' := V \cup Y$, $Y \cap V = \emptyset$, $|X| = |Y|$, and there is a bijection b from X to Y, and $E' := E \cup b$.

5.2.2. Proofs

It is easy to see that a completion of a workflow graph is again a workflow graph.

Corollary 5.2.1 (Completion is a workflow graph)**.** *A completion of a prefix of a workflow graph is a workflow graph.*

Corollary 5.2.1 follows immediately from its definition.

Split Miner produces models with OR-joins and then applies Algorithm 9 to replace trivial OR-joins with AND- and XOR-joins. Next, we demonstrate that an acyclic workflow graph in which all joins are OR-joins is guaranteed to be sound.

Lemma 5.2.1 (Sound acycic workflow graphs)**.** *If a workflow graph $G := (V, E, l)$ is acyclic, i.e., E is irreflexive, such that for every split $v \in V$ it holds that $l(v) \neq OR$ and for every join $v \in V$ it holds that $l(v) = OR$, then G is sound.*

Proof. (Sketch) By Noetherian induction on prefixes of G, we show that a completion of G is sound. Let $\mathscr{G}$ be the set of all prefixes of G. Then, $(\mathscr{G}, \sqsubseteq)$ is a well-founded set.

- **Induction basis:** A completion of $(s, \emptyset, \emptyset)$, where $s \in V$ is the source of G, is sound. Clearly, a workflow graph $(\{s,x\}, \{(s,x)\}, \emptyset)$, where $s \neq x$, is sound.
- **Induction step:** Let $G' := (V', E', l') \in \mathcal{G}$ be a prefix of G. Assume that for every $G'' \in \mathcal{G}$ such that $G'' \sqsubset G'$ it holds that a completion of G'' is sound. Let $\hat{G} := (\hat{V}, \hat{E}, \hat{l}) \sqsubset G'$ be such that $E' \setminus \hat{E} = \{e\}$, $e \in E$. We distinguish these two cases:
    1. The target node v of e is in $\hat{V}$. Then, v is a join for which it holds that $l(v) = OR$; indeed, for every join $j \in V$ it holds that $l(j) = OR$. By definition of the semantics of workflow graphs, refer to Definition 5.2.2, and because G and, thus, G' are acyclic, it holds that a completion of G' is sound. The only interesting case here is when for the source v' of e it holds that $l(v') = XOR$ and $|v' \bullet| > 1$. In this case, one cannot reach a deadlock or unsafe state from a state s of a completion of G' for which $s(e) = 1$ because for every join j that can be reached from v' via a directed path it holds that $l(j) = OR$.
    2. The target node v of e is not in $\hat{V}$. Then, v is not a join and, clearly, a completion of G' is sound. Note that a deadlock or unsafe reachable marking in a workflow graph can be introduced only via a fresh join.

Hence, a completion of G is sound. It is easy to see that if a completion of G is sound, then G is also sound.

Before showing that a replacement of a trivial OR-join preserves soundness of an acyclic workflow graph, we define this transformation formally.

Definition 5.2.4 (Replacement of OR join)**.** *Let $G := (V, E, l)$ be a workflow graph, such that for every split $v \in V$ it holds that $l(v) \neq OR$. Let $v \in V$ be a join, such that $l(v) = OR$. Let s be the result of Algorithm 9 for the input of G and v. The result of* replacement *of v in G is a workflow graph $G' := (V, E, l')$, where $l' := \{(x,y) \in l \mid x \neq v\} \cup \{(v,s)\}$*

Next, we demonstrate that replacement of an OR-join in a sound acyclic workflow graph results in a sound workflow graph.

Lemma 5.2.2 (Replacement of OR join)**.** *Let $G := (V, E, l)$ be a sound acyclic workflow graph such that for every split $v \in V$ it holds that $l(v) \neq OR$. Let $v \in V$ be a join, such that $l(v) = OR$. If $G' := (V, E, l')$ is the result of replacement of v in G, then G' is sound.*

Proof. (Sketch) By definition of Algorithm 9. We distinguish three cases:

- ○ $l'(v) = OR$. It holds that $G' = G$ and, thus G' is sound.
- ○ $l'(v) = AND$. According to Algorithm 9, it holds that for all the splits on all the paths from the minimal dominator d of v to v are AND-splits. Also, it holds that no two distinct outgoing edges of a split on a path from v to d lead to the same incoming edge of v. Hence, it holds that from every reachable

state that marks an incoming edge of v one can reach a state that marks all the incoming edges of v. Hence, the sets of all the reachable states of G and G' are the same. Thus, G' is sound.

- ○ $l'(v) = XOR$. According to Algorithm 9, it holds that for all the splits on all the paths from the minimal dominator d of v to v are XOR-splits. Also, it holds that no two distinct outgoing edges of a split on a path from v to d lead to the same incoming edge of v. Hence, it holds that from every reachable state that marks exactly one incoming edge of v one cannot reach a state that marks two incoming edges of v. Hence, the sets of all the reachable states of G and G' are the same. Thus, G' is sound.

Clearly, one can apply replacements to all the OR-joins to obtain a sound acyclic workflow graph without trivial OR-joins.

Theorem 5.2.1. *Replacement of OR joinsLet* $G := (V,E,l)$ *be a sound acyclic workflow graph such that for every split* $v \in V$ *it holds that* $l(v) \neq OR$. *Let* $f : V' \rightarrow \{XOR, AND, OR\}$, *where* V' *is the set of all joins* v *of* G *for which it holds that* $l(v) = OR$, *and* $f(v)$, $v \in V'$, *is the result of Algorithm 9 for the input of* G *and* v. *If* $G' := (V,E,l')$, *where* $l' := \{(x,y) \in l \mid y \neq OR\} \cup \{(x, f(x)) \in V' \times \{XOR, AND, OR\} \mid x \in V'\}$, *then* G' *is sound.*

The proof of Theorem 5.2.1 follows immediately from Lemma 5.2.2 and the observation that the order of replacements of OR joins in a sound acyclic workflow graph does not influence the result. The latter fact holds (i) because G and G' have the same structure, i.e., the same nodes and edges, and (ii) because replacements of OR joins preserve the semantics of splits, i.e., for every $v \in V$ such that $|v\bullet| > 1$ it holds that $l(v) = l'(v)$; note that the result of Algorithm 9 depends only on these two factors.

5.3. Evaluation

We implemented Split Miner (hereafter SM) as a standalone Java application.[8] The tool takes as input an event log in MXML or XES format and the values for the thresholds ε and η, and it outputs a BPMN process model. Using this implementation, we plugged SM in our benchmark (introduced in Chapter 4) and we empirically compared SM against the state-of-the-art APDAs. The dataset and the experimental setup we used to assess SM are the same we used in our benchmark, while for the hyper-parameter optimization of SM we used steps of 0.10 for both its filtering thresholds (η) and its parallelism threshold (ε). In the following, we report the benchmark results, now including SM, and we discuss them. However, given that we already extensively discussed the results of α\$, IM, ETM, FO, S-HM$_6$, and HILP in Chapter 4, in this section we mostly focus on

---

[8] Available at `http://apromore.org/platform/tools`

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity			Sound	Exec. Time(s)
		Fitness	Precision	F-score		Size	CFC	Struct.		
BPIC12	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	**0.98**	0.50	0.66	**0.98**	**59**	37	**1.00**	yes	6.60
	ETM	0.44	**0.82**	0.57	t/o	67	**16**	**1.00**	yes	14,400
	FO	-	-	-	-	102	117	0.13	no	9.66
	S-HM$_6$	-	-	-	-	88	46	0.40	no	227.80
	HILP	-	-	-	-	300	460	-	no	772.20
	SM	0.97	0.72	**0.83**	0.97	63	43	0.73	yes	**0.58**
BPIC13$_{cp}$	$\alpha\$$	-	-	-	-	18	9	-	no	10,112.60
	IM	0.82	**1.00**	0.90	0.82	**9**	4	**1.00**	yes	0.10
	ETM	**1.00**	0.70	0.82	t/o	38	38	**1.00**	yes	14,400
	FO	-	-	-	-	25	23	0.60	no	0.06
	S-HM$_6$	0.94	0.99	**0.97**	0.94	15	6	**1.00**	yes	130.0
	HILP	-	-	-	-	10	**3**	-	yes	0.10
	SM	0.99	0.93	0.96	**0.99**	13	7	**1.00**	yes	**0.03**
BPIC13$_{inc}$	$\alpha\$$	0.35	0.91	0.51	t/o	15	7	0.47	yes	4,243.14
	IM	0.92	0.54	0.68	0.92	13	7	**1.00**	yes	1.00
	ETM	**1.00**	0.51	0.68	t/o	32	144	**1.00**	yes	14,400
	FO	-	-	-	-	43	54	0.77	no	1.41
	S-HM$_6$	0.91	**0.96**	0.93	0.91	**9**	**4**	**1.00**	yes	0.80
	HILP	-	-	-	-	24	9	-	yes	2.50
	SM	0.98	0.92	**0.95**	**0.98**	15	10	**1.00**	yes	**0.23**
BPIC14$_f$	$\alpha\$$	0.47	0.63	0.54	t/o	62	36	0.31	yes	14,057.48
	IM	**0.89**	0.64	0.74	**0.89**	31	18	**1.00**	yes	3.40
	ETM	0.61	**1.00**	0.76	t/o	**23**	**9**	**1.00**	yes	14,400
	FO	-	-	-	-	37	46	0.38	no	27.73
	S-HM$_6$	-	-	-	-	202	132	0.73	no	147.40
	HILP	-	-	-	-	80	59	-	no	7.30
	SM	0.77	0.91	**0.84**	0.78	24	15	**1.00**	yes	**0.59**
BPIC15$_{1f}$	$\alpha\$$	0.71	0.76	0.73	t/o	219	91	0.22	yes	3,545.9
	IM	0.97	0.57	0.71	**0.96**	164	108	**1.00**	yes	0.60
	ETM	0.56	**0.94**	0.70	t/o	**67**	**19**	**1.00**	yes	14,400
	FO	**1.00**	0.76	0.87	0.94	146	91	0.25	yes	1.02
	S-HM$_6$	-	-	-	-	204	116	0.56	no	128.10
	HILP	-	-	-	-	282	322	-	no	4.40
	SM	0.90	0.88	**0.89**	0.89	114	43	0.48	yes	**0.48**
BPIC15$_{2f}$	$\alpha\$$	-	-	-	-	348	164	0.08	no	8,787.48
	IM	0.93	0.56	0.70	0.94	193	123	**1.00**	yes	0.70
	ETM	0.62	**0.91**	0.74	t/o	**95**	**32**	**1.00**	yes	14,400
	FO	-	-	-	-	195	159	0.09	no	0.61
	S-HM$_6$	**0.98**	0.59	0.74	**0.97**	259	150	0.29	yes	163.2
	HILP	-	-	-	-	-	-	-	-	t/o
	SM	0.77	0.90	**0.83**	0.75	124	41	0.32	yes	**0.25**

Table 17: Split Miner default parameters evaluation results for the public logs – Part 1/2.

SM results and how it compares against the other APDAs, and specially those that showed to be above average, such as: IM and ETM, as well as FO, and S-HM$_6$.

5.3.1. Evaluation Results

Default Parameters. The results of the default parameters evaluation are shown in Tables 17, 18, 19, and 20. As in Chapter 4, also in this case we used "-" to report that a given accuracy or complexity measurement could not be reliably obtained due to syntactical or behavioral issues in the discovered model (i.e., a discon-

Log	Discovery Method	Accuracy			Gen.	Complexity			Sound?	Exec.
		Fitness	Precision	F-score	(3-Fold)	Size	CFC	Struct.		Time (sec)
BPIC15$_{\mathrm{3f}}$	$\alpha\$$	-	-	-	-	319	169	0.03	no	10,118.15
	IM	**0.95**	0.55	0.70	**0.95**	159	108	**1.00**	yes	1.30
	ETM	0.68	0.88	0.76	t/o	**84**	**29**	**1.00**	yes	14,400
	FO	-	-	-	-	174	164	0.06	no	0.89
	S-HM$_6$	**0.95**	0.67	0.79	**0.95**	159	151	0.13	yes	139.90
	HILP	-	-	-	-	433	829	-	no	1,062.90
	SM	0.78	**0.94**	**0.85**	0.78	92	**29**	0.61	yes	**0.36**
BPIC15$_{\mathrm{4f}}$	$\alpha\$$	-	-	-	-	272	128	0.13	no	6,410.25
	IM	0.96	0.58	0.73	0.96	162	111	**1.00**	yes	0.7
	ETM	0.65	**0.93**	0.77	t/o	**83**	**28**	**1.00**	yes	14,400
	FO	-	-	-	-	157	127	0.14	no	0.50
	S-HM$_6$	**0.99**	0.64	0.78	**0.99**	209	137	0.37	yes	136.90
	HILP	-	-	-	-	364	593	-	no	14.7
	SM	0.73	0.91	**0.81**	0.74	98	31	0.31	yes	**0.25**
BPIC15$_{\mathrm{5f}}$	$\alpha\$$	0.62	0.75	0.68	t/o	280	126	0.10	yes	7,603.19
	IM	0.94	0.18	0.30	0.94	134	95	**1.00**	yes	1.50
	ETM	0.57	**0.94**	0.71	t/o	**88**	**18**	**1.00**	yes	14,400
	FO	**1.00**	0.71	0.83	**1.00**	166	125	0.15	yes	0.56
	S-HM$_6$	**1.00**	0.70	0.82	**1.00**	211	135	0.35	yes	141.90
	HILP	-	-	-	-	-	-	-	-	t/o
	SM	0.79	**0.94**	**0.86**	0.78	105	30	0.33	yes	**0.27**
BPIC17$_{\mathrm{f}}$	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	**0.98**	0.70	0.82	**0.98**	**35**	20	**1.00**	yes	13.30
	ETM	0.76	**1.00**	0.86	t/o	42	**4**	**1.00**	yes	14,400
	FO	-	-	-	-	98	82	0.25	no	64.33
	S-HM$_6$	0.95	0.62	0.75	0.94	42	13	0.97	yes	143.20
	HILP	-	-	-	-	222	330	-	no	384.50
	SM	0.96	0.81	**0.88**	0.96	39	21	**1.00**	yes	**2.53**
RTFMP	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	0.99	0.70	0.82	0.99	34	20	**1.00**	yes	10.90
	ETM	0.99	0.92	0.95	t/o	57	32	**1.00**	yes	14,400
	FO	**1.00**	0.94	0.97	0.97	31	32	0.19	yes	2.57
	S-HM$_6$	0.98	0.95	0.96	0.98	163	97	**1.00**	yes	262.70
	HILP	-	-	-	-	57	53	-	no	3.50
	SM	**1.00**	**0.97**	**1.00**	**1.00**	**25**	**18**	0.40	yes	**1.25**
SEPSIS	$\alpha\$$	-	-	-	-	146	156	0.01	no	3,883.12
	IM	**0.99**	0.45	0.62	**0.96**	50	32	**1.00**	yes	0.40
	ETM	0.83	0.66	0.74	t/o	108	101	**1.00**	yes	14,400
	FO	-	-	-	-	60	63	0.28	no	0.17
	S-HM$_6$	0.92	0.42	0.58	0.92	279	198	**1.00**	yes	242.70
	HILP	-	-	-	-	87	129	-	no	1.60
	SM	0.76	**0.77**	**0.77**	0.77	**39**	**25**	0.82	yes	**0.05**

Table 18: Split Miner default parameters evaluation results for the public logs – Part 2/2.

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity			Sound	Exec. Time(s)
		Fitness	**Precision**	**F-score**		**Size**	**CFC**	**Struct.**		
PRT1	$\alpha\$$	-	-	-	t/o	45	34	-	no	11,168.54
	IM	0.90	0.67	0.77	0.90	**20**	**9**	**1.00**	yes	2.08
	ETM	**0.99**	0.81	0.89	t/o	23	12	**1.00**	yes	14,400
	FO	-	-	-	-	30	28	0.53	no	0.95
	S-HM$_6$	0.88	0.77	0.82	0.88	59	39	**1.00**	yes	122.16
	HILP	-	-	-	-	195	271	-	no	2.59
	SM	0.98	**0.99**	**0.98**	**0.98**	27	16	**1.00**	yes	**0.47**
PRT2	$\alpha\$$	-	-	-	-	134	113	0.25	no	3,438.72
	IM	ex	ex	ex	ex	45	33	**1.00**	yes	1.41
	ETM	0.57	**0.94**	0.71	t/o	86	**21**	**1.00**	yes	14,400
	FO	-	-	-	-	76	74	0.59	no	0.88
	S-HM$_6$	-	-	-	-	67	105	0.43	no	1.77
	HILP	-	-	-	-	190	299	-	no	21.33
	SM	**0.81**	0.70	**0.75**	**0.81**	**38**	28	0.87	yes	**0.31**
PRT3	$\alpha\$$	0.67	0.76	0.71	0.67	70	40	0.11	yes	220.11
	IM	0.98	0.68	0.80	0.98	37	20	**1.00**	yes	0.44
	ETM	0.98	0.86	**0.92**	t/o	51	37	**1.00**	yes	14,400
	FO	**1.00**	0.86	**0.92**	**1.00**	34	37	0.32	yes	0.50
	S-HM$_6$	**1.00**	0.83	0.91	**1.00**	40	38	0.43	yes	0.67
	HILP	-	-	-	-	343	525	-	no	0.73
	SM	0.82	**0.92**	0.87	0.84	**29**	**13**	0.76	yes	**0.17**
PRT4	$\alpha\$$	0.86	0.93	0.90	t/o	**21**	**10**	**1.00**	yes	13,586.48
	IM	0.93	0.75	0.83	0.93	27	13	**1.00**	yes	1.33
	ETM	0.84	0.85	0.84	t/o	64	28	**1.00**	yes	14,400
	FO	-	-	-	-	37	40	0.54	no	6.33
	S-HM$_6$	**1.00**	0.86	**0.93**	**1.00**	370	274	**1.00**	yes	241.57
	HILP	-	-	-	-	213	306	-	no	5.31
	SM	0.83	**1.00**	0.91	0.88	31	19	0.77	yes	**0.45**
PRT5	$\alpha\$$	1.00	1.00	1.00	1.00	10	1	1.00	yes	2.02
	IM	1.00	1.00	1.00	1.00	10	1	1.00	yes	0.03
	ETM	1.00	1.00	1.00	1.00	10	1	1.00	yes	2.49
	FO	1.00	1.00	1.00	1.00	10	1	1.00	yes	**0.02**
	S-HM$_6$	1.00	1.00	1.00	1.00	10	1	1.00	yes	0.11
	HILP	1.00	1.00	1.00	1.00	10	1	1.00	yes	0.05
	SM	1.00	1.00	1.00	1.00	10	1	1.00	yes	**0.02**
PRT6	$\alpha\$$	0.80	0.77	0.79	0.80	38	17	0.24	yes	40.10
	IM	0.99	0.82	0.90	0.99	23	10	**1.00**	yes	2.30
	ETM	0.98	0.80	0.88	t/o	41	16	**1.00**	yes	14,400
	FO	**1.00**	0.91	0.95	**1.00**	22	17	0.41	yes	0.05
	S-HM$_6$	**1.00**	0.91	0.95	**1.00**	22	17	0.41	yes	0.42
	HILP	-	-	-	-	157	214	-	no	0.13
	SM	0.94	**1.00**	**0.97**	0.94	**15**	**4**	**1.00**	yes	**0.02**

Table 19: Split Miner default parameters evaluation results for the proprietary logs - Part 1/2.

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity			Sound?	Exec. Time (sec)
		Fitness	Precision	F-score		Size	CFC	Struct.		
PRT7	$\alpha\$$	0.85	0.90	0.88	0.85	29	**9**	0.48	yes	143.66
	IM	**1.00**	0.73	0.84	**1.00**	29	13	**1.00**	yes	0.13
	ETM	0.90	0.81	0.85	t/o	60	29	**1.00**	yes	14,400
	FO	0.99	**1.00**	0.99	0.99	**26**	16	0.39	yes	0.08
	S-HM$_6$	**1.00**	**1.00**	**1.00**	**1.00**	163	76	**1.00**	yes	249.74
	HILP	-	-	-	-	278	355	-	no	0.27
	SM	0.91	**1.00**	0.95	0.92	29	10	0.48	yes	**0.06**
PRT8	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	**0.98**	0.33	0.49	**0.93**	111	92	**1.00**	yes	**0.41**
	ETM	0.35	**0.88**	0.50	t/o	**75**	**12**	**1.00**	yes	14,400
	FO	-	-	-	-	228	179	0.74	no	0.55
	S-HM$_6$	-	-	-	-	388	323	0.87	no	370.66
	HILP	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	SM	0.97	0.41	**0.57**	0.93	241	322	0.82	yes	1.28
PRT9	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	0.90	0.61	0.73	0.89	28	16	**1.00**	yes	63.70
	ETM	0.75	0.49	0.59	0.74	**27**	**13**	**1.00**	yes	1,266.71
	FO	-	-	-	-	32	45	0.72	no	42.83
	S-HM$_6$	**0.96**	0.98	**0.97**	**0.96**	723	558	**1.00**	yes	318.69
	HILP	-	-	-	-	164	257	-	no	51.47
	SM	0.92	**1.00**	0.96	0.92	29	19	**1.00**	yes	**9.11**
PRT10	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	0.96	0.79	0.87	0.96	**41**	**29**	**1.00**	yes	2.50
	ETM	**1.00**	0.63	0.77	t/o	61	45	**1.00**	yes	14,400
	FO	0.99	0.93	**0.96**	**0.99**	52	85	0.64	yes	0.98
	S-HM$_6$	-	-	-	-	77	110	-	no	1.81
	HILP	-	-	-	-	846	3130	-	no	2.55
	SM	0.97	**0.95**	**0.96**	0.97	60	49	0.75	yes	**0.47**
PRT11	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	t/o	t/o	t/o	t/o	549	365	**1.00**	yes	121.50
	ETM	**0.10**	**1.00**	**0.18**	t/o	**21**	**3**	**1.00**	yes	14,400
	FO	-	-	-	-	680	713	0.68	no	81.33
	S-HM$_6$	ex	ex	ex	ex	ex	ex	ex	ex	ex
	HILP	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	SM	-	-	-	-	712	609	0.12	no	**19.53**
PRT12	$\alpha\$$	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	IM	**1.00**	0.77	0.87	**1.00**	32	25	**1.00**	yes	3.94
	ETM	0.63	**1.00**	0.77	t/o	**21**	**8**	**1.00**	yes	14,400
	FO	-	-	-	-	87	129	0.38	no	1.67
	S-HM$_6$	-	-	-	-	4370	3191	**1.00**	yes	347.57
	HILP	-	-	-	-	926	2492	-	no	7.34
	SM	0.96	0.97	**0.97**	0.96	78	65	0.78	yes	**0.36**

Table 20: Split Miner default parameters evaluation results for the proprietary logs - Part 2/2.

Log	Discovery Method	Accuracy			Gen.	Complexity			Exec.
		Fitness	Precision	F-score	(3-Fold)	Size	CFC	Struct.	Time (sec)
Frequency Absolute Best	$\alpha\$$	0	0	0	0	1	2	0	0
	IM	**9**	1	0	**9**	5	2	**24**	1
	ETM	5	**11**	2	0	**10**	**13**	**24**	0
	FO	4	1	2	4	1	0	0	0
	S-HM$_6$	**9**	2	4	**9**	1	1	8	0
	HILP	0	0	0	0	0	1	0	0
	SM	2	10	**17**	5	4	4	7	**23**
Frequency Second Best	$\alpha\$$	0	3	0	0	1	1	1	0
	IM	8	2	3	**9**	**9**	**13**	0	6
	ETM	3	4	**7**	0	1	2	0	0
	FO	2	2	5	2	4	0	6	**17**
	S-HM$_6$	1	4	6	2	1	1	7	1
	HILP	0	0	0	0	1	0	0	0
	SM	**10**	**9**	3	8	**9**	7	**11**	0
Total	$\alpha\$$	0	3	0	0	2	3	1	0
	IM	**17**	3	3	**18**	**14**	**15**	**24**	7
	ETM	8	15	9	0	11	**15**	**24**	0
	FO	6	3	7	6	5	0	6	17
	S-HM$_6$	10	6	10	11	2	2	15	1
	HILP	0	0	0	0	1	1	0	0
	SM	12	**19**	**20**	13	13	11	18	**23**

Table 21: Best score frequencies for each quality dimension (default parameters evaluation, inc. Split Miner).

nected model or an unsound model). Additionally, to report the occurrence of a timeout or an exception during the execution of a discovery method we used "*t/o*" and "*ex*", respectively. We highlighted in bold the best score for each measure on each log, we underlined the second-best score, and we summarized these achievements in Table 21.

SM systematically performed very good across the whole evaluation. It showed to have the capabilities to excel in fitness, precision, or generalization, and it performed the best on F-score. At the same time, SM achieved often the highest (or second highest) simplicity with its discovered process models. While IM remained the best APDA in terms of discovering fitting process models, it fell behind SM in terms of precision and F-score. ETM and SM achieved respectively 19 and 21 times a precision greater than 0.80, and scored the highest precision 11 and 10 times (respectively). However, as highlighted already in our benchmark results, ETM scores high precision at the cost of lower fitness. SM, not being affected by this drawback, was able to maintain balanced high scores of fitness and precision, which allowed it to stand out for its F-score. SM achieved 20 times out of 24 an F-score above 0.80, outperforming all the other approaches 17 times, and being the runner up for 3 times.

To strengthen our conclusions we used the Mann-Whitney U-test [6] to statistically validate the following hypothesis: (i) IM significantly outperforms ETM in terms of fitness (U=148, $n_1 = 22$, $n_2 = 24$, Z=2.55, P<0.02); (ii) IM significantly outperforms SM in terms of fitness (U=150.5, $n_1 = 22$, $n_2 = 23$, Z=2.33, P<0.02); (iii) ETM significantly outperforms IM in terms of precision (U=98.5,

$n_1 = 22$, $n_2 = 23$, Z=3.64, P<0.01); (iv) SM significantly outperforms IM in terms of precision (U=68, $n_1 = 22$, $n_2 = 23$, Z=4.20, P<0.01); (v) SM significantly outperforms IM in terms of F-score (U=88.5, $n_1 = 22$, $n_2 = 23$, Z=3.74, P<0.01); (vi) SM significantly outperforms ETM in terms of F-score (U=113, $n_1 = 22$, $n_2 = 23$, Z=3.47, P<0.01). Such results summarise our discussion above showing that: IM excels in fitness, ETM and SM in precision, and SM in F-score.

In terms of complexity, SM joined the outstanding results of IM and ETM (see Table 21). SM discovered 7 times fully block-structured models (structuredness equal to 1.00), and 11 times highly structured models, being just behind IM and ETM, that discover fully block-structured models by design. SM also placed itself ahead of S-HM$_6$, which strives for discovering block-structured models. ETM and SM discovered the smallest or second-smallest model for more than 50% of the times. These models also had low CFC, and were the ones with the lowest CFC 13 times (ETM) and four times (SM).

On the execution time, SM was the clear winner. It outperformed all the other approaches, regardless of the input. It was the fastest APDA 23 times out of 24, discovering a model in less than a second for 19 logs. In contrast, ETM was the slowest approach, reaching the timeout of four hours for 22 logs, highlighting that the quality of its discovered process models is paid in execution time.

Despite the remarkable results of SM, we remind that it does not guarantee soundness (except in the case of acyclic process models), as a consequence, it produced one unsound model out of 24 (from the PRT11 log). We note that, for the remaining 23 logs, SM was able to discover sound models, some of which containing loops. This latter result shows that SM is well ahead than other approaches that do not guarantee soundness, such as FO and S-HM$_6$. Indeed, by comparison, FO discovered only 8 sound models, while S-HM$_6$ discovered 18 sound models.

Hyper-parameter Optimization. The results of the hyper-parameter optimization evaluation are shown in Tables 22–25. We marked with a "*" the discovery approaches that were not able to complete the exploration of the solution space within 24 hours of timeout time. The purpose of this second evaluation was to understand whether the F-score of the models discovered by SM can be improved even further when fine-tuning its input parameters, what price SM pays if such an improvement materialises (i.e. at the cost of which other quality dimension), and how its results compare to the other APDAs.

In line with our goal, Tables 22 and 23 report the accuracy and complexity scores of the discovered models with the highest F-score. SM remained the most accurate APDA in precision (12 times out of 23), followed by IM (being it the best 8 times). Also, SM confirmed to be the most balanced discovery approach, scoring the highest F-score most of the times (18 out of 23), while discovering very simple models along with IM, the latter delivered 14 times the simplest model (for size, CFC and structuredness), followed by SM (six times). The results for generalization show SM achieving the best generalization eight times, followed

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity		
		Fitness	Precision	F-score		Size	CFC	Struct.
BPIC12	IM	0.90	0.69	0.78	0.91	69	46	**1.00**
	FO*	**1.00**	0.07	0.14	-	112	1369	**1.00**
	S-HM$_6$	0.96	0.67	0.79	0.96	97	110	0.63
	SM	0.97	**0.72**	**0.83**	**0.97**	**63**	**43**	0.73
BPIC13$_{cp}$	IM	**0.99**	**0.98**	**0.98**	**0.99**	**11**	**5**	**1.00**
	FO	0.00	0.42	0.00	-	19	16	**1.00**
	S-HM$_6$	0.96	0.92	0.94	0.96	20	14	0.80
	SM	**0.99**	0.93	0.96	**0.99**	13	7	**1.00**
BPIC13$_{inc}$	IM	0.90	0.87	0.89	0.90	**13**	**5**	**1.00**
	FO	0.00	0.36	0.00	-	42	76	0.88
	S-HM$_6$	0.93	**0.98**	**0.96**	0.93	16	10	**1.00**
	SM	**0.98**	0.92	0.95	**0.98**	15	10	**1.00**
BPIC14$_f$	IM	0.75	**0.97**	0.85	0.75	**19**	**4**	**1.00**
	FO	**0.97**	0.81	**0.88**	0.31	27	34	0.56
	S-HM$_6$	0.91	0.84	**0.88**	**0.91**	178	117	0.97
	SM	0.85	0.86	0.86	0.85	30	22	0.70
BPIC15$_{1f}$	IM	0.81	0.68	0.74	0.83	140	70	**1.00**
	FO	**1.00**	0.76	0.87	0.94	146	91	0.26
	S-HM$_6$	0.88	**0.89**	0.89	0.58	1576	550	**1.00**
	SM	0.95	0.86	**0.90**	**0.95**	**122**	**51**	0.45
BPIC15$_{2f}$	IM	0.71	0.76	0.74	0.69	**141**	61	**1.00**
	FO	**0.99**	0.63	0.77	**0.99**	195	164	0.09
	S-HM$_6$	**0.99**	0.62	0.76	**0.99**	246	167	0.19
	SM	0.81	**0.86**	**0.83**	0.81	**141**	**58**	0.31
BPIC15$_{3f}$	IM	0.65	**0.99**	0.79	0.63	**73**	**8**	**1.00**
	FO	**0.99**	0.60	0.75	**0.99**	162	163	0.07
	S-HM$_6$	0.81	0.77	0.79	0.81	231	77	0.97
	SM	0.78	0.94	**0.85**	0.78	92	29	0.61
BPIC15$_{4f}$	IM	0.73	0.84	0.78	0.75	108	42	**1.00**
	FO	**1.00**	0.67	0.80	**1.00**	155	128	0.14
	S-HM$_6$	0.99	0.66	0.79	0.99	217	145	0.36
	SM	0.77	**0.90**	**0.83**	0.78	**102**	**35**	0.34
BPIC15$_{5f}$	IM	0.64	0.88	0.74	0.65	**105**	**34**	**1.00**
	FO	**1.00**	0.71	0.83	**1.00**	166	125	0.15
	S-HM$_6$	0.82	**0.94**	0.87	0.81	610	166	0.96
	SM	0.84	0.92	**0.88**	0.82	108	36	0.22
BPIC17$_f$	IM	**1.00**	0.70	0.82	**1.00**	39	24	**1.00**
	FO*	-	-	-	-	-	-	-
	S-HM$_6$	0.97	0.70	0.81	0.97	51	25	**1.00**
	SM	0.94	**0.83**	**0.88**	0.94	**37**	**19**	**1.00**
RTFMP	IM	0.94	0.98	0.96	0.94	28	**10**	**1.00**
	FO	**1.00**	0.94	0.97	0.84	31	32	0.19
	S-HM$_6$	0.95	**0.99**	0.97	0.95	82	30	**1.00**
	SM	**1.00**	0.97	**0.98**	**1.00**	**25**	18	0.40
SEPSIS	IM	0.62	**0.98**	0.76	0.76	**31**	**14**	**1.00**
	FO	**0.96**	0.36	0.53	0.30	51	109	0.33
	S-HM$_6$	0.80	0.39	0.52	**0.86**	299	187	**1.00**
	SM	0.76	0.77	**0.77**	0.77	39	25	0.82

Table 22: Scores of the models with the best F-score discovered with hyper-parameter optimization (public logs), inc. Split Miner.

Log	Discovery Method	Accuracy			Gen. (3-Fold)	Complexity		
		Fitness	**Precision**	**F-score**		**Size**	**CFC**	**Struct.**
PRT1	IM	0.91	0.89	0.90	0.91	**24**	**11**	**1.00**
	FO	**0.98**	0.92	0.95	**0.99**	25	29	0.72
	S-HM$_6$	0.95	0.97	0.96	0.95	37	29	0.92
	SM	**0.98**	**0.98**	**0.98**	0.98	27	16	**1.00**
PRT2	IM	-	-	-	-	-	-	-
	FO	**1.00**	0.17	0.30	**1.00**	55	241	**0.93**
	S-HM$_6$	-	-	-	-	-	-	-
	SM	0.81	**0.70**	**0.75**	0.81	**38**	**28**	0.87
PRT3	IM	0.87	**0.93**	0.90	0.87	**27**	**8**	**1.00**
	FO	**1.00**	0.86	**0.92**	**1.00**	34	37	0.32
	S-HM$_6$	0.99	0.85	0.91	0.96	40	34	0.48
	SM	0.95	0.89	**0.92**	0.95	33	24	0.55
PRT4	IM	0.86	**1.00**	0.92	0.86	**21**	**5**	**1.00**
	FO	**1.00**	0.87	0.93	-	32	41	0.50
	S-HM$_6$	0.93	0.96	**0.95**	0.93	66	55	0.77
	SM	0.97	0.93	**0.95**	**0.97**	36	32	0.56
PRT5	IM	1.00	1.00	1.00	**1.00**	12	1	**1.00**
	FO	1.00	1.00	1.00	0.95	**10**	1	**1.00**
	S-HM$_6$	1.00	1.00	1.00	**1.00**	12	1	**1.00**
	SM	1.00	1.00	1.00	**1.00**	**10**	1	**1.00**
PRT6	IM	0.90	**1.00**	0.95	0.90	17	**2**	**1.00**
	FO	**1.00**	0.91	0.95	0.96	22	17	0.41
	S-HM$_6$	0.98	0.96	**0.97**	**0.98**	24	15	0.46
	SM	0.94	**1.00**	**0.97**	0.94	**15**	4	**1.00**
PRT7	IM	0.88	**1.00**	0.93	0.88	**23**	**5**	**1.00**
	FO	0.99	**1.00**	0.99	0.99	26	16	0.39
	S-HM$_6$	**1.00**	**1.00**	**1.00**	**1.00**	165	76	**1.00**
	SM	0.93	**1.00**	0.96	0.92	34	16	0.29
PRT8	IM*	**1.00**	0.09	0.16	**0.99**	**95**	**86**	**1.00**
	FO	-	-	-	-	-	-	-
	S-HM$_6$	0.93	0.42	0.58	0.89	221	422	0.83
	SM	0.77	**0.58**	**0.66**	0.76	214	176	0.93
PRT9	IM	0.93	0.71	0.80	0.93	**28**	**14**	**1.00**
	FO	-	-	-	-	-	-	-
	S-HM$_6$	**0.99**	0.99	**0.99**	**0.99**	41	59	0.68
	SM	**0.99**	**1.00**	**0.99**	**0.99**	41	34	0.68
PRT10	IM	**1.00**	0.81	0.89	**1.00**	**47**	**33**	**1.00**
	FO	0.99	0.93	0.96	-	52	85	0.64
	S-HM$_6$	0.98	0.83	0.90	0.98	1440	972	**1.00**
	SM	0.98	**0.95**	**0.97**	0.98	64	55	0.66
PRT12	IM	0.93	0.92	0.93	0.93	**37**	**26**	**1.00**
	FO	**1.00**	0.80	0.89	0.94	60	237	0.87
	S-HM$_6$	0.88	0.67	0.76	0.88	3943	2314	**1.00**
	SM	0.97	**0.97**	**0.97**	**0.97**	80	75	0.76

Table 23: Scores of the models with the best F-score discovered with hyper-parameter optimization (proprietary logs), inc. Split Miner.

Metric	Discovery Method	BPIC Logs										RTFMP	SEPSIS
		12	13_{cp}	13_{inc}	14_{f}	15_{1f}	15_{2f}	15_{3f}	15_{4f}	15_{5f}	17_{f}		
Fitness	IM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	**1.00**	0.00	0.00	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	-	**1.00**	**1.00**
	S-HM$_6$	0.96	**1.00**	0.93	**1.00**	**1.00**	0.99	0.99	**1.00**	**1.00**	**1.00**	0.95	0.80
	SM	0.98	**1.00**	**1.00**	0.85	0.97	0.95	0.81	0.83	0.92	0.96	**1.00**	0.98
Prec.	IM	**0.92**	**1.00**	0.89	**1.00**	**0.97**	**1.00**	**0.99**	0.91	**0.99**	**0.89**	0.98	**0.98**
	FO	0.07	0.42	0.36	0.81	0.76	0.63	0.60	0.67	0.71	-	0.94	0.36
	S-HM$_6$	0.67	0.92	**0.98**	0.84	0.89	0.78	0.77	0.66	0.94	0.70	0.99	0.39
	SM	0.80	0.93	0.92	0.91	0.92	0.92	0.94	**0.93**	0.96	0.83	**1.00**	0.81
F-score	IM	0.78	**0.98**	0.89	0.85	0.74	0.74	0.79	0.78	0.74	0.82	0.96	0.76
	FO	0.14	0.00	0.00	**0.88**	0.87	0.77	0.75	0.80	0.83	-	0.97	0.53
	S-HM$_6$	0.79	0.94	**0.96**	**0.88**	0.89	0.76	0.79	0.79	0.87	0.81	0.97	0.52
	SM	**0.83**	0.96	0.95	0.86	**0.90**	**0.83**	**0.85**	**0.83**	**0.88**	**0.88**	**0.98**	**0.77**
Gen. (3-Fold)	IM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	-	-	0.00	0.96	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	-	0.90	0.94
	S-HM$_6$	0.96	0.99	0.93	**1.00**	**1.00**	0.99	0.99	**1.00**	**1.00**	**1.00**	0.95	0.86
	SM	0.98	**1.00**	**1.00**	0.85	0.97	0.94	0.82	0.85	0.91	0.96	**1.00**	0.98
Size	IM	**15**	9	**9**	17	**63**	**30**	**17**	**25**	**32**	23	**11**	**14**
	FO	112	19	39	27	113	137	114	111	117	-	23	40
	S-HM$_6$	87	**8**	16	**13**	74	246	207	139	232	**22**	82	299
	SM	53	13	15	23	107	117	92	92	103	36	22	36
CFC	IM	**8**	2	**3**	2	10	**7**	**8**	**9**	**9**	6	**5**	**9**
	FO	1369	16	54	34	47	57	53	46	44	-	13	35
	S-HM$_6$	65	**0**	10	**0**	**0**	167	77	35	140	**0**	30	187
	SM	28	7	10	15	35	34	28	27	28	19	10	21
Struct.	IM	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.29	**1.00**	**1.00**	**1.00**	-	**1.00**	**1.00**
	S-HM$_6$	0.77	**1.00**	**1.00**	0.97	**1.00**	0.99	0.98	0.37	0.96	**1.00**	**1.00**	**1.00**
	SM	**1.00**	**1.00**	**1.00**	**1.00**	0.48	0.41	0.62	0.35	0.33	**1.00**	0.54	0.92

Table 24: Best scores achieved in hyper-parameter evaluation by each approach on each quality dimension (public logs), inc. Split Miner.

Metric	Discovery Method	Proprietary (PRT) Logs										
		# 1	**# 2**	**# 3**	**# 4**	**# 5**	**# 6**	**# 7**	**# 8**	**# 9**	**# 10**	**# 12**
Fitness	IM	**1.00**	-	**1.00**	**1.00**	1.00	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**1.00**	**1.00**	-	-	**1.00**	**1.00**
	S-HM$_6$	**1.00**	-	**1.00**	**1.00**	1.00	**1.00**	**1.00**	**1.00**	0.99	**1.00**	0.88
	SM	**1.00**	0.83	**1.00**	0.97	1.00	0.94	0.93	0.99	0.99	**1.00**	**1.00**
Prec.	IM	0.89	-	**0.93**	**1.00**	1.00	**1.00**	**1.00**	0.09	0.89	0.95	**0.99**
	FO	0.92	0.17	0.89	0.94	1.00	0.91	**1.00**	-	-	**0.96**	0.80
	S-HM$_6$	0.97	-	0.85	0.96	1.00	0.96	**1.00**	0.42	0.99	0.83	0.67
	SM	**0.98**	**0.71**	0.92	**1.00**	1.00	**1.00**	**1.00**	**0.58**	**1.00**	0.95	0.98
F-score	IM	0.90	-	0.90	0.92	1.00	0.95	0.93	0.16	0.80	0.89	0.93
	FO	0.95	0.30	**0.92**	0.93	1.00	0.95	0.99	-	-	0.96	0.89
	S-HM$_6$	0.96	-	0.91	**0.95**	1.00	**0.97**	**1.00**	0.58	0.99	0.90	0.76
	SM	**0.98**	**0.75**	**0.92**	**0.95**	1.00	**0.97**	0.96	**0.66**	**0.99**	**0.97**	**0.97**
Gen. (3-Fold)	IM	**1.00**	-	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**
	FO	**1.00**	**1.00**	**1.00**	-	0.95	0.96	**1.00**	-	-	-	0.94
	S-HM$_6$	**1.00**	-	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.98	0.99	**1.00**	0.88
	SM	**1.00**	0.83	**1.00**	0.97	**1.00**	0.94	0.92	0.94	0.99	**1.00**	0.99
Size	IM	14	-	**27**	**21**	12	17	23	95	**16**	27	**25**
	FO	22	55	30	30	10	22	**21**	-	-	48	60
	S-HM$_6$	**13**	-	37	66	12	24	40	**59**	41	**23**	3943
	SM	27	**33**	29	29	**8**	**13**	**21**	210	29	52	66
CFC	IM	4	-	**8**	**5**	1	**2**	**5**	86	**2**	10	**18**
	FO	16	74	24	33	1	17	7	-	-	44	66
	S-HM$_6$	**0**	-	27	55	1	15	15	**0**	58	**0**	2314
	SM	16	**24**	13	18	**0**	3	6	176	19	39	49
Struct.	IM	**1.00**	-	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	FO	0.77	0.93	0.57	0.54	**1.00**	0.41	0.81	-	-	0.81	0.87
	S-HM$_6$	**1.00**	-	0.72	**1.00**	**1.00**	0.50	**1.00**	0.85	**1.00**	**1.00**	**1.00**
	SM	**1.00**	**1.00**	0.76	0.77	**1.00**	**1.00**	0.76	0.93	**1.00**	0.89	0.90

Table 25: Best scores achieved in hyper-parameter evaluation by each approach on each quality dimension (proprietary logs), inc. Split Miner.

very closely by FO (seven times) and by S-HM$_6$ (six times), with IM achieving the highest generalization in four cases only.

Finally, Tables 24 and 25 report the best score that each discovery approach can achieve in each dimension.[9] We note that SM is not always able to maximally optimize fitness, regardless of the input parameters chosen its fitness scores are in the range 0.80-1.00. Similar are the results for generalization, with FO and IM leading, and S-HM$_6$ and SM following. As for precision, SM strikes better results, being always in the range 0.90-1.00 (excluding the outliers: PRT2 and PRT8). Finally, in F-score SM distinguished itself again, with scores in the range 0.80-1.00 and often over 0.90 (more than 50% of the times), while S-HM$_6$ is the runner up with a similar performance. As for model simplicity, IM retains its leading role, followed by SM.

To conclude, the hyper-parameter optimization evaluation showed that SM is able to push even further the quality of its models, similarly to the other APDAs. However, SM reaches the upper limits of fitness and generalization earlier than the upper limits of precision and F-score, indicating that SM is more likely to discover precise models rather than general ones, as opposed to IM, that tends to achieve higher fitness and generalisation.

5.4. Summary

In this chapter, we addressed our **RQ2** [10] presenting a new APDA, namely Split Miner, whose underlying algorithms were designed to discover process models from event logs in an effective and efficient manner, striving for the best trade-off between the four quality dimensions: fitness, precision, generalization, and soundness.

Split Miner operates over six steps: *DFG and loops discovery*; *concurrency discovery*; *filtering*; *splits discovery*; *joins discovery*; and *OR-joins minimization*. Each of these steps plays a fundamental role in the discovery of a highly fitting and precise process model, that is at the same time simple and deadlock-free. About the latter property, we formally proved that Split Miner guarantees to discover sound acyclic process models, and deadlock-free cyclic process models. Differently than other APDAs that guarantee soundness, Split Miner does not achieve such a result enforcing full structuredness on its discovered process models (as opposed to IM or ETM), making SM the very first[11] APDA that guarantees to discover sound (or deadlock-free) process models that are not fully structured. Furthermore, we showed that the time complexity of Split Miner is polynomial, guaranteeing efficiency in terms of execution times during the process discovery.

---

[9] Full results of the hyper-parameter optimization evaluation are included in the zip file available at `https://doi.org/10.5281/zenodo.1219321`

[10] *How to strike a trade-off between the various quality dimensions in automated process discovery in an* effective *and* efficient *manner?*

[11] According to our literature review and benchmark, respectively Chapter 3 and 4.

Finally, we empirically evaluated SM using our benchmark. As we expected from the theoretical grounds, the results of the evaluation highlighted that SM performs better than the state-of-the-art APDAs, achieving higher F-scores, balancing fitness and precision, and maintaining a low complexity in its discovered process models. The empirical evaluation also allowed us to note that despite SM cannot guarantee soundness for cyclic process models, SM is yet able to discover sound cyclic process models. Lastly, SM proved to be the fastest APDA, discovering process models in less than a second regardless of the input event logs.

To conclude, the algorithms we presented in this chapter, which combined together form Split Miner, allowed us to strike a trade-off between the various quality dimensions in automated process discovery in an *effective* and *efficient* manner.

6. MARKOVIAN ACCURACY

The results of our benchmark (see Chapter 2) showed that the input parameters of an APDA play an important role in the discovery of the process model and ultimately on its quality. In particular, the hyper-parameter optimization evaluation highlighted that all the existing APDAs can achieve better results when their input parameters are finely tuned. However, such a tuning is expensive in terms of computational power and time, sometimes requiring up to 24 hours to discover a process model as opposed to seconds (or minutes) when choosing default parameters. This inefficiency is the direct consequence of two different problems: (i) low scalability of the accuracy measures, especially precision; and (ii) a blind exploration of the solution space. Addressing both problems is our strategy to tackle our **RQ3**[1]. In this chapter, we focus on the former problem.

Several measures of fitness and precision have been proposed and empirically evaluated in the literature [145]. Existing precision measures (and to a lesser extent also fitness measures) suffer from scalability issues when applied to models discovered from real-life event logs. In addition, Tax et al. [5] have shown that none of the existing precision measures fulfils a set of five intuitive properties, namely *precision axioms*. These axioms were recently revised by Syring et al. [4], who also proposed a set of desirable properties of fitness measures, namely *fitness propositions*, showing that only the latest fitness and precision measures designed by Polyvyanyy et al. [146] can fulfil the proposed properties.

In this chapter, [2] we present a family of fitness and precision measures based on the idea of comparing the k^{th}-order Markovian abstraction of a process model against that of an event log. We show that the proposed fitness and precision measures can fulfil all aforementioned properties for a suitable k dependent on the log. In other words, when measuring fitness and precision, we do not need to explore the entire state space of a process model behaviour but only its state space up to a certain memory horizon, reducing in this way the computational power and time required.

We empirically evaluate exemplars of the proposed family of measures using: (i) a synthetic collection of models and logs previously used to assess the suitability of precision measures (applicable also to fitness measures); and (ii) a set of models discovered from 20 real-life event logs using IM, SM, and SHM.

The chapter is structured as follows. Section 6.1 gives an overview of the state-of-the-art fitness and precision measures, and examines them with respect to the theoretical properties postulated by Tax et al. [5] and van der Aalst [147]. Section 6.2 describes the Markovian abstractions and how to generate them starting from a process model and an event log. Section 6.3 defines the fitness and precision measures based on the Markovian abstractions, and shows that these

---

[1] *How can the accuracy of an automated process discovery approach be efficiently optimized?*

[2] Corresponding to [19, 20].

measures fulfil the theoretical properties. Finally, Section 6.4 reports the empirical evaluation, and Section 6.5 summarises the chapter.

6.1. Fitness and Precision in Automated Process Discovery

In this section, we provide an overview of existing measures of fitness and precision. We then introduce a collection of theoretical properties for fitness and precision measures introduced in previous research studies, and we discuss the limitations of existing fitness and precision measures with respect to these properties.

6.1.1. Fitness Measures

One of the earliest and simplest measures of fitness is *token-based fitness* [148]. Given an event log and a process model, represented as a Workflow net[3], token-based fitness is equal to one minus a normalized count of the number of tokens that need to be added or deleted when replaying every trace of the event log against the Workflow net. If while replaying a trace, we reach a state (a.k.a. *marking*) where the next event in the trace does not match any enabled transition, then tokens are added to the Workflow net in order to enable (and fire) the transition corresponding to the event. When we finish replaying a trace, if any tokens are left behind in any place other than the end place of the Workflow net, these tokens are deleted. This fitness measure was successively extended by De Medeiros [149] in the context of automated process discovery using genetic algorithms, proposing two variants: the *continuous parsing* and the *improved continuous semantics fitness*, which optimize the computational efficiency of token-based fitness at the cost of exactness. Another variant of token-based fitness was proposed by vanden Broucke et al. [150]. The idea of this latter approach is to decompose the Workflow net into single-entry single-exit regions and to analyse each region independently, so as to increase scalability and to provide a more localised feedback. While computationally efficient, especially with the aforementioned optimizations, token-based fitness measures have been criticized as being arbitrary, because they make "local decisions" when deciding how to fix a replay error.

Adriansyah et al. [151] argue that a more robust approach to measuring fitness is to compute, for each trace in the log, the closest trace recognized by the process model, and to measure fitness based on the minimal number of error-corrections required to align these two traces (a.k.a. *minimal alignment cost*). This observation underpins the *alignment-based fitness* measure [151], which is equal to one minus a normalized sum of the minimal alignment cost between each trace in the log and the closest corresponding trace recognized by the model. While conceptually sound, *alignment-based fitness* measures sometimes suffer from scalability

---

[3] A Workflow net is a Petri net with one single start place, one single end place, and such that every transition is on a path from the start to the end place.

issues when applied to real-life event logs, due to the need to compute the optimal alignment between the process model and each trace in the log.

In a recent study, Leemans et al. [152] proposed the *projected conformance checking (PCC) fitness* measure. The idea of *PCC fitness* is to construct an automaton from the process model (namely A_m) and from the event log (A_l). To control the size of these automata, the *PCC fitness* focuses on a subset of the activities in the event log, by ignoring the less frequent activities. The two automata A_m and A_l are then used to generate a third automaton, namely $A_{l,m}$, capturing their common behavior. The fitness value is then computed as the ratio between the number of outgoing edges of each state in $A_{l,m}$ and the number of outgoing edges of the corresponding state(s) in A_l. As we will observe in Section 6.4 , *PCC fitness* suffers from similar (and sometimes more pronounced) scalability issues as alignment-based fitness.

Lastly, Polyvyanyy et al. [146] proposed a fitness measure based on the concept of *topological entropy* [153] (i.e. *Entropy-based fitness, EBF*). Topological entropy can be used to compute the cardinality of an irreducible language. In our context, the behavior recorded in an event log (or captured in a process model) can be considered as a language defined over the alphabet represented by the set of the process activity labels. The cardinality of the event log (or process model) language would correspond to the (non necessarily finite) number of distinct traces recorded in the event log (or observed in the process model). Therefore, Polyvyanyy et al. propose to compute fitness as the ratio of the topological entropy of the intersection of the event log and the process languages over the topological entropy of the event log language. This is done encoding the languages as deterministic finite automata (DFA), short-circuiting the DFA in order to turn the encoded languages irreducible, and using the DFA adjacency matrixes to compute the topological entropy of each language [153].

6.1.2. Precision Measures

Greco et al. [154] define the precision of a (model, log) pair as the *set difference* (*SD*) between the set of traces recognized by the process model and the set of traces in the event log. This measure is a direct operationalization of the concept of precision, but it is not applicable to models with cycles since the latter have an infinite set of traces.

Rozinat and van der Aalst [155] proposed the *advanced behavioral appropriateness* (*ABA*) precision measure. *ABA* precision is based on the comparison between the sets of activity pairs that sometimes but not always follow each other, and the set of activity pairs that sometimes but not always precede each other. The comparison is performed on the sets of activity pairs extracted both from the model and the log behaviors. The *ABA* precision measure does not scale to large models and it is undefined for process models without concurrency or conflict relations [5].

De Weerdt et al. [156] proposed the *negative events* precision measure (*NE*). This method works by inserting inexistent (so-called negative) events to enhance the traces in the log. A negative event is inserted after a given prefix of a trace if this event is never observed preceded by that prefix anywhere in the log. The traces extended with negative events are then replayed on the model. If the model can parse some of the negative events, it means that the model has additional behavior. This approach is however heuristic: it does not guarantee that all additional behavior is identified.

Muñoz-Gama and Carmona [157] proposed the *escaping edges* (*ETC*) precision measure. This approach starts by building a *prefix automaton* from the event log. It then replays the process model behavior on top of this prefix automation and counts the number of times that the model can perform a move that the prefix automaton cannot. Each of these mismatches is called an *escaping edge*. The original *ETC* precision measure assumes that the process model perfectly fits the log. An extension of *ETC* precision applicable to logs containing non-fitting traces, namely *alignments-based ETC* precision (herein ETC_a), in proposed in [158]. *ETC* and ETC_a are not very accurate, since they only count the escaping edges without taking into account how much the process model behavior diverges from the log behavior after each escaping edge.

More recently, van Dongen et al. [2] proposed the *anti-alignment* precision (*AA*). This measure analyses the anti-alignments of the process model behavior to assess the model's precision. An anti-alignment of length n is a trace in the process model behavior of length at most equal to n, which maximizes the Levenshtein distance from all traces in the log. The major drawback of *AA* precision is that it cannot be applied in a real-life context due to its scalability issues and the difficulties in tuning its input parameters [2].

Alongside the *PCC fitness* measure outlined above, Leemans et al. [152] proposed a dual measure of precision, namely *PCC precision*. This latter measure is computed in a similar way as *PCC fitness*, with the difference that *PCC precision* is the ratio between the number of outgoing edges of each state in $A_{l,m}$ and the number of outgoing edges of the corresponding states occurring in A_m.

As for the case of PCC, alongside the *Entropy-based fitness* (EBF), Polyvyanyy et al. [146] proposed a precision measure based on the concept of *topological entropy* [153]. To compute the *Entropy-based precision* (EBP), the authors propose to use the ratio of the topological entropy of the intersection of the event log and the process languages over the topological entropy of the process language.

We note that *PCC* and *Entropy-based conformance checking* (ECC) are the only of the above approaches where fitness and precision measures are computed based on the same abstraction of the model and the log behavior, specifically automata-based abstractions. We follow up on this idea by proposing fitness and precision measures based on a unified behavioral abstraction, but designed to be more scalable than *PCC* and *ECC*, yet fulfilling a set of desirable theoretical properties for fitness and precision measures, exposed in the following.

6.1.3. Fitness Propositions

Syring et al. [4, 147] proposed seven propositions to capture intuitive properties behind the concept of fitness in automated process discovery. Before introducing these propositions, we formally define the notions of process model behavior and event log behavior, as well as the auxiliary concepts of trace and sub-trace.

Definition 6.1.1. ***[Trace]*** *Given a set of activity labels Ω, we define a trace on Ω as a sequence $\tau_\Omega = \langle t_1, t_2, \dots, t_{n-1}, t_n \rangle$, such that $\forall 1 \leq i \leq n, t_i \in \Omega$.* [4] *Furthermore, we denote with τ_i the activity label in position i, and we use the symbol Γ_Ω to refer to the universe of traces on Ω. With abuse of notation, hereinafter we refer to any $t \in \Omega$ as an activity instead of an activity label.*

Definition 6.1.2. ***[Subtrace]*** *Given a trace $\tau = \langle t_1, t_2, \dots, t_{n-1}, t_n \rangle$, with the notation $\tau^{i \to j}$, we refer to the subtrace $\langle\, t_i, t_{i+1}, \dots, t_{j-1}, t_j \,\rangle$, where $0 < i < j \leq n$. We extend the subset operator to traces, i.e., given two traces τ and $\widehat{\tau}$, $\widehat{\tau}$ is contained in τ, shorthanded as $\widehat{\tau} \subset \tau$, if and only if (iff) $\exists i, j \in \mathbb{N} \mid \tau^{i \to j} = \widehat{\tau}$.*

Definition 6.1.3. ***[Process Model Behavior]*** *Given a process model P (regardless of its representation) and being Ω the set of its activities. We refer to the model behavior as $\mathscr{B}_P \subseteq \Gamma_\Omega$, where $\forall \langle t_1, t_2, \dots, t_{n-1}, t_n \rangle \in \mathscr{B}_P$ there exists an execution of P that allows to execute the sequence of activities $\langle t_1, t_2, \dots, t_{n-1}, t_n \rangle$, where t_1 is the first activity executed, and t_n the last.* [5]

Definition 6.1.4. ***[Event Log and Event Log Behavior]*** *Given a set of activities Ω, an event log L is a finite multiset of traces defined over Ω. The event log behavior of L is defined as $\mathscr{B}_L = support(L)$.*[6]

Given the above definitions, the seven fitness propositions are spelled out below.

Definition 6.1.5. ***[Fitness Propositions]***

- ***Proposition-1F.*** *A fitness measure is a deterministic function $fit : \mathscr{L} \times \mathscr{P} \rightarrow [0,1] \cap \mathbb{R}$, where $\mathscr{L}$ is the universe of event logs, and $\mathscr{P}$ is the universe of processes.*
- ***Proposition-2F.*** *Given two process models P_1, P_2 and a log L, if the behavior of P_1 is equal to the behavior of P_2, the fitness value of P_1 must be equal to the fitness value of P_2. Formally, if $\mathscr{B}_{P_1} = \mathscr{B}_{P_2} \Longrightarrow fit(L, P_1) = fit(L, P_2)$.*
- ***Proposition-3.*** *Given two process models P_1, P_2 and a log L, if the behavior of P_1 is contained in the behavior of P_2, the fitness value of P_2 must be equal to or greater than the fitness value of P_1. Formally, if $\mathscr{B}_{P_1} \subseteq \mathscr{B}_{P_2} \Longrightarrow fit(L, P_2) \geq fit(L, P_1)$.*

---

[4] For the sake of simplicity, we refer to τ_Ω as τ where there is no ambiguity.

[5] When $\mathscr{B}_P = \Gamma_\Omega$, we say that P is a *flower model*. Intuitively, a flower model is a process model that recognizes any trace consisting of any number of occurrences of a given set of tasks, in any order.

[6] The support of a multiset is the set of distinct elements of the multiset.

Fitness Measure		**Propositions**						
Name	**Label**	**1F**	**2F**	**3**	**4**	**5**	**6**	**7**
Token-based fitness	*Token*	×	×	×	√	×	√	√
Alignment-based fitness	*Align*	√	√	√	√	×	√	√
PCC fitness	PCC_f	√	√	√	√	×	√	√
Entropy-based fitness	*EBF*	√	√	√	√	√	√	√

Table 26: Fitness propositions fulfiled by existing fitness measures (according to [4]).

- ***Proposition-4.*** *Given a process model P and two event logs* L_1, L_2*, if the behavior of* L_2 *is contained in the behavior of P, the fitness value of the model measured over the union of* L_1 *and* L_2 *must be equal to or greater than the fitness value measured over* L_1*. Formally, if* $\mathscr{B}_{L_2} \subseteq \mathscr{B}_P \Longrightarrow \mathit{fit}(L_1 \cup L_2, P) \geq \mathit{fit}(L_1, P)$.
- ***Proposition-5.*** *Given a process model P and two event logs* L_1, L_2*, if none of the behavior of* L_2 *is contained in the behavior of P, the fitness value of the model measured over* L_1 *must be equal to or greater than the fitness value measured over the union of* L_1 *and* L_2*. Formally, if* $\mathscr{B}_{L_2} \cap \mathscr{B}_P = \varnothing \Longrightarrow \mathit{fit}(L_1, P) \geq \mathit{fit}(L_1 \cup L_2, P)$.
- ***Proposition-6.*** *Given a process model P and an event log L, altering the frequencies of the traces recorded in L without altering their distribution should not alter the fitness value. Formally, let* $n \in \mathbb{N}$ *and* $L^n = \bigcup_n L \Longrightarrow \mathit{fit}(L, P) = \mathit{fit}(L^n, P)$.
- ***Proposition-7.*** *Given a process model P and a log L, if the behavior of L is contained in the behavior of P, the fitness value of P must be equal to 1. Formally, if* $\mathscr{B}_L \subseteq \mathscr{B}_P \Longrightarrow \mathit{fit}(L, P) = 1$.

Syring et al. [4] also assessed the existing fitness measures against these propositions, we report in Table 26 the assessment results.

6.1.4. Precision Propositions

Syring et al. [4, 147] proposed also eight propositions to capture intuitive properties behind the concept of precision in automated process discovery. These propositions extend and generalise the axioms proposed by Tax et al. [5], though ruling out Axiom-3, which instead we integrate in the following.[7]

Definition 6.1.6. ***[Precision Propositions]***

- ***Proposition-1P.*** *A precision measure is a deterministic function* $prec : \mathscr{L} \times \mathscr{P} \rightarrow \mathbb{R}$*, where* $\mathscr{L}$ *is the universe of event logs, and* $\mathscr{P}$ *is the universe of processes.*
- ***Proposition-2P.*** *Given two process models* P_1, P_2 *and a log L, if the behavior of* P_1 *is equal to the behavior of* P_2*, the precision value of* P_1 *must be equal*

---

[7] We kept the same numbering for propositions and axioms as in the former studies [4, 5].

Precision Measure		**Propositions**								
Name	**Label**	**1P = A1**	**2P = A4**	**8 ⊃ A2**	**9 = A5**	**10**	**11**	**12**	**13**	**A3**
Set Difference	*SD*	×	√	√	√	√	√	√	√	×
Advanced Behavioral Appropriateness	*ABA*	×	×	×	√	×	√	√	√	×
Negative Events	*NE*	×	×	×	×	×	√	√	√	?
Alignment-based ETC (one-align)	*ETC*	×	×	×	×	×	√	×	×	×
PCC precision	PCC_p	√	√	×	×	×	√	√	√	?
Anti-alignment	*AA*	√	√	√	×	×	√	√	√	?
Entropy-based Precision	*EBP*	√	√	√	√	√	√	√	√	?

Table 27: Precision propositions fulfiled by existing precision measures (according to [4]), including the Axioms 1-5 (A1-A5) as discussed in [5].

to the precision value of P_2*. Formally, if* $\mathscr{B}_{P_1} = \mathscr{B}_{P_2} \Longrightarrow prec(L,P_1) = prec(L,P_2)$*.*

- ***Proposition-8.*** *Given two process models* P_1, P_2 *and a log L, if the behavior of* P_1 *is contained in the behavior of* P_2*, and the intersection between the behavior of* log *and the behavior of* P_2 *not contained in the behavior of* P_1 *is empty, the precision value of* P_1 *must be equal to or greater than the precision value of* P_2*. Formally, if* $\mathscr{B}_{P_1} \subseteq \mathscr{B}_{P_2}$ *and* $\mathscr{B}_L \cap (\mathscr{B}_{P_2} \setminus \mathscr{B}_{P_1}) = \varnothing \Longrightarrow prec(L,P_1) \geq prec(L,P_2)$*.*
- ***Proposition-9.*** *Given a process model P and two event logs* L_1, L_2*, if the behavior of* L_1 *is contained in the behavior of* L_2*, and the behavior of* L_2 *is contained in the behavior of P, the precision value of the model measured over* L_2 *must be equal to or greater than the precision value measured over* L_1*. Formally, if* $\mathscr{B}_{L_1} \subseteq \mathscr{B}_{L_2} \subseteq \mathscr{B}_P \Longrightarrow prec(L_2,P) \geq prec(L_1,P)$*.*
- ***Proposition-10.*** *Given a process model P and two event logs* L_1, L_2*, if the behavior of* L_1 *is contained in the behavior of* L_2*, and none of the behavior of* L_2 *not contained in the behavior of* L_1 *can is contained in the behavior of P, the precision value of P over* L_1 *must be equal to the precision value of P over* L_2*. Formally, if* $\mathscr{B}_{L_1} \subseteq \mathscr{B}_{L_2}$ *and* $\mathscr{B}_P \cap (\mathscr{B}_{L_2} \setminus \mathscr{B}_{L_1}) = \varnothing \Longrightarrow prec(L_1,P) = prec(L_2,P)$*.*
- ***Proposition-11.*** *Given a process model P and an event log L, altering the frequencies of the traces recorded in L without altering their distribution should not alter the precision value. Formally, let* $n \in \mathbb{N}$ *and* $L^n = \bigcup_n L \Longrightarrow prec(L,P) = prec(L^n,P)$*.*
- ***Proposition-12.*** *Given a process model P and a log L, if the behavior of P is equal to the behavior of L, the precision value of P must be equal to 1. Formally, if* $\mathscr{B}_P = \mathscr{B}_L \Longrightarrow prec(L,P) = 1$*.*
- ***Proposition-13.*** *Given a process model P and a log L, if the behavior of P is contained (or equal) in the behavior of L, the precision value of P must be equal to 1. Formally, if* $\mathscr{B}_P \subseteq \mathscr{B}_L \Longrightarrow prec(L,P) = 1$*.*
- ***Axiom-3.*** *Given two process models* P_1, P_2 *and a log L, if the behavior of L is contained in the behavior of* P_1*, and* P_2 *is the flower model, the precision value of* P_1 *must be greater than the precision value of* P_2*. Formally, if*

$\mathscr{B}_L \subseteq \mathscr{B}_{P_1} \subset \mathscr{B}_{P_2} = \Gamma_\Omega \Longrightarrow prec(L,P_1) > prec(L,P_2)$.

Table 26 shows which precision measures fulfil which propositions according to [4], what precision measures fulfil Axiom-3 as discussed in [5], and the relations between propositions and axioms.

6.2. K^{th}-order Markovian Abstraction

A k$^{\text{th}}$-order Markovian abstraction (M$^{\text{k}}$-abstraction) is a graph composed of a set of states (S) and a set of edges ($E \subseteq S \times S$). In an M$^{\text{k}}$-abstraction, every state $s \in S$ represents a (sub)trace of at most length k, e.g. $s = \langle b,c,d \rangle$, and every state of an M$^{\text{k}}$-abstraction is unique, i.e. there are no two states representing the same (sub)trace. Two states $s_1, s_2 \in S$ are connected via an edge $e = (s_1, s_2) \in E$ iff s_1 and s_2 satisfy the following three properties: i) the first activity of the (sub)trace represented by s_1 can occur before the (sub)trace represented by s_2, ii) the last activity of the (sub)trace represented by s_2 can occur after the (sub)trace represented by s_1, and iii) the two (sub)traces represented by s_1 and s_2 overlap with the exception of their first and last activity, e.g. $e = (\langle b,c,d \rangle, \langle c,d,e \rangle)$. An M$^{\text{k}}$-abstraction is defined w.r.t. a given order k, which defines the length of the (sub)traces encoded in the states. An M$^{\text{k}}$-abstraction contains a special state (denoted as $-$) representing the source and sink of the M$^{\text{k}}$-abstraction. Intuitively, every state represents either a *trace* of length less than or equal to k or a *subtrace* of length k, whilst every edge represents an existing *subtrace* of length $k+1$ or a *trace* of length less than or equal to $k+1$. Thus, M$^{\text{k}}$-abstraction captures how all the traces of the input behavior evolve in chunks of length k. The definitions below show the construction of an M$^{\text{k}}$-abstraction from a given $\mathscr{B}_X$.

Definition 6.2.1. ***[kth-order Markovian Abstraction]*** *Given a set of traces $\mathscr{B}_X$, the k-order Markovian Abstraction is the graph $M^k_X = (S,E)$ where S is the set of states and $E \subseteq S \times S$ is the set of edges, such that*

- $S = \{-\} \cup \{\tau \,:\, \tau \in \mathscr{B}_X \wedge |\tau| \leq k\} \cup \{\tau^{i\to j} \,:\, \tau \in \mathscr{B}_X \wedge |\tau| > k \wedge \left|\tau^{i\to j}\right| = k\}$
- $E = \{(-,\tau) \,:\, \tau \in S \wedge |\tau| \leq k\} \cup \{(\tau,-) \,:\, \tau \in S \wedge |\tau| \leq k\} \cup \{(-,\tau) : \exists \widehat{\tau} \in \mathscr{B}_X \; s.t. \; \tau = \widehat{\tau}^{1\to k}\} \cup \{(\tau,-) : \exists \widehat{\tau} \in \mathscr{B}_X \; s.t. \; \tau = \widehat{\tau}^{(|\widehat{\tau}|-k+1)\to|\widehat{\tau}|}\} \cup \{(\tau',\tau'') \,:\, \tau',\tau'' \in S \wedge \tau' \oplus \tau''_{|\tau''|} = \tau'_1 \oplus \tau'' \wedge \exists \widehat{\tau} \in \mathscr{B}_X \; s.t. \; \tau'_1 \oplus \tau'' \subseteq \widehat{\tau}\}$ [8]

A fundamental property of Markovian abstractions (valid for any order k) is the following.

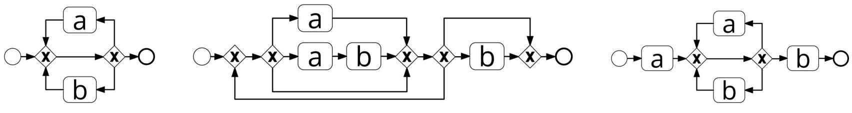
Theorem 6.2.1. ***[Equality and Containment Inheritance]*** *Given two sets of traces $\mathscr{B}_X$ and $\mathscr{B}_Y$, and their respective M^k-abstractions $M^k_X = (S_X, E_X)$ and $M^k_Y = (S_Y, E_Y)$, any equality or containment relation between $\mathscr{B}_X$ and $\mathscr{B}_Y$ is inherited by E_X and E_Y. Formally, if $\mathscr{B}_X = \mathscr{B}_Y$ then $E_X = E_Y$, or if $\mathscr{B}_X \subset \mathscr{B}_Y$ then $E_X \subseteq E_Y$.*

---

[8] The operator $\oplus$ is the *concatenation* operator.

Traces
$\langle a,a,b\rangle$
$\langle a,b,b\rangle$
$\langle a,b,a,b,a,b\rangle$

Table 28: Log L^*.



(a) Flower Proc. (b) Process Y (c) Process X

Figure 22: Examples of processes in the BPMN language.

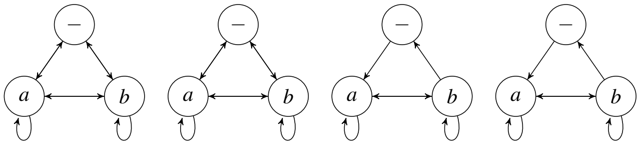
Proof. (Sketch) This follows by construction. Specifically, every edge $e \in E_X$ represents either a subtrace $\tau^{x \to y} : \tau \in \mathscr{B}_X \ \wedge \ |\tau^{x \to y}| = k+1$, or a trace $\tau : \tau \in \mathscr{B}_X \ \wedge \ |\tau| < k+1$. It follows that from the same sets of traces the corresponding M$^{\mathrm{k}}$-abstractions contain the same sets of edges.

It should be noted that nothing can be said about traces in $\mathscr{B}_Y \setminus \mathscr{B}_X$, i.e. adding new traces to $\mathscr{B}_X$ does not imply that new edges are added to E_X. As a result the relation $\mathscr{B}_X \subset \mathscr{B}_Y$ only guarantees $E_X \subseteq E_Y$.

Moreover, an M^1-abstraction is equivalent to a *directly-follows graph* (used as starting point by many process discovery approaches [8, 12, 159], including Split Miner). Instead, when k approaches infinity then M$^\infty$-abstraction is equivalent to listing all the traces. The order of a Markovian abstraction, i.e. k, allows us to play with the level of behavioral approximation. For example, let us consider the event log L^* as in Tab. 28, and the Process-X (P_x) in Fig. 22c. Their respective M^1-abstractions: $M^1_{L^*}$ and $M^1_{P_x}$ are shown in Fig. 23d and 23c. We can observe that $M^1_{L^*} = M^1_{P_x}$, though $\mathscr{B}_{P_x}$ is infinite whilst $\mathscr{B}_{L^*}$ is not.

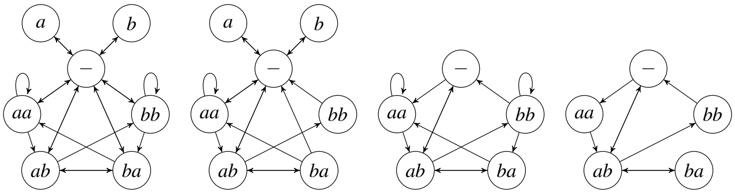
This is an example of how the M^1-abstraction can over-approximate the original behavior. Increasing k reduces the over-approximation, thus allowing us to detect more behavioral differences, e.g. increasing k to 2, the differences between L^* and P_x emerge as shown in Figs. 24d and 24c. We note that for k equal to the length of the longest trace in the log, the Markovian abstraction of the log is exact. A similar statement cannot be made for the Markovian abstraction of the model, since the longest trace of a model may be infinite.

Markovian Property. We note that the term *Markovian* is traditionally used to describe something relating to a Markov process. However, in its essence, the Markovian property simply declares that given the current state of a process, its future state is independent of its past, i.e. independent of how the current state was reached [160]. The adjective *Markovian* in our behavioural abstractions recalls this general property characterising all the Markov models, indeed, given a state of our Markovian abstraction, the future state is independent of the previ-



(a) (b) (c) (d)

Figure 23: From left to right: the M^1-abstraction of the Flower Process, Process-Y, Process-X and the event log L^*.



(a) (b) (c) (d)

Figure 24: From left to right, the M^2-abstraction of the Flower Process, Process-Y, Process-X and the event log L^*.

ously traversed ones. Finally, we remark that while our Markovian abstraction is not intended to be a Markov model, it could be turned into one by adding the corresponding probabilities to its edges. However, while in the case of a Markovian abstraction generated from an event log these probabilities could be extracted from the event log, the same could not be seamlessly achieved for Markovian abstractions generated from process models. In fact, process models are usually not annotated with probabilities.

6.2.1. Generating the M^k-abstraction of an Event Log

Given an event log, it is always possible to generate its M^k-abstraction in polynomial time, since the log behavior is a finite set of traces. Algorithm 10 shows how we build the M^k-abstraction abstraction given as inputs: an event log L and the order k. First, we create the set of states (S) and the sets of edges (E) of the M^k-abstraction (lines 1 and 2). Then, we initialize the source/sink state $s_0 = -$. Finally, we iterate over all the traces recorded in the log as follows. For each trace τ with length less or equal to k, we add τ to S, and two new edges (s_0, τ) and (τ, s_0), lines 7 to 9. For each trace τ with length greater than k, we read the trace using a sliding window of size k, moving this latter one activity forward per iteration (lines 15 to 18). The sliding window (in Algorithm 10, depicted by the state s_w) is initialised as the prefix of τ, and the edge (s_0, s_w) is added to the set E (lines 11 to 13). Then, at each iteration, s_w slides over τ, adding a new state to the set S and a new edge to the set E. When s_w becomes the suffix of τ, the iteration is over and a final edge is added to the set E (line 19).

Algorithm 10: Calculating $\mathrm{M^k}$-abstraction of an Event Log

```
input  : Event Log $L$
input  : Order $k$
Result: $\mathrm{M^k}$-abstraction $M_L^k$

Set $S \leftarrow \varnothing$;
Set $E \leftarrow \varnothing$;
State $s_0 \leftarrow -$;
add $s_0$ to $S$;

for $\tau \in L$ do
    if $|\tau| \leq k$ then
        add $\tau$ to $S$;
        add $(s_0, \tau)$ to $E$;
        add $(\tau, s_0)$ to $E$;
    else
        State $s_w \leftarrow \tau^{1 \to k}$;
        add $s_w$ to $S$;
        add $(s_0, s_w)$ to $E$;
        for $i \in [1, |\tau| - k]$ do
            State $s_x \leftarrow s_w$;
            $s_w \leftarrow \tau^{(1+i) \to (k+i)}$;
            add $s_w$ to $S$;
            add $(s_x, s_w)$ to $E$;
        add $(s_w, s_0)$ to $E$;

$M_L^k \leftarrow (S, E)$;
return $M_L^k$
```

Time Complexity. Algorithm 10 iterates over each activity of each trace of the input event log L. Consequently, the time complexity is polynomial on the total number of activities recorded in the event log, $O\left(\sum_{\tau \in L} |\tau|\right)$.

6.2.2. Generating the $\mathrm{M^k}$-abstraction of a Process

Given a process P, its behavior $\mathscr{B}_P$ can be finite or infinite. While in theory, for processes with finite behavior we could use Algorithm 10 this will not work for processes with infinite behaviour. To address this problem, we represent $\mathscr{B}_P$ as a behavioral automaton (Definition 6.2.2), and we replay this latter to generate the $\mathrm{M^k}$-abstraction of the process.

Definition 6.2.2. ***[Behavioral Automaton of a Process]*** *Given a process P, its behavioral automaton is a graph $R = (N, A)$, where N is the set of nodes and $A \subseteq N \times N \times \Omega$ is the set of arcs. A node $n \in N$ represents a Process Execution State (PES), whilst an arc $(n_1, n_2, t) \in A$ represents the possibility to move from PES n_1 to the PES n_2 via the execution of activity t. Furthermore, we define $n_0 \in N$ as the initial PES, such that: $\forall (n_1, n_2, t) \in A \Rightarrow n_2 \neq n_0$.*

Algorithm 11 shows the steps required to generate the $\mathrm{M^k}$-abstraction for a given process P. First, we initialize the set S and the set E of the $\mathrm{M^k}$-abstraction, and we add the source/sink state $s_0 = -$ to S. Then, we generate the behavioral automaton of the process (R), we retrieve its initial state (n_0), and we initialize all the necessary data structures to perform its replay as follows (lines 5 to 16). We create a queue Q and we add n_0 to it. This queue stores the PESs that have to be explored. We create a map $\mathscr{V}$, to store for each PES the set of (sub)traces whose execution led to the PES and progression needs to be explored (all these sets are initialized as empty, see line 13). We create a map $\mathscr{X}$, to store for each PES the set of (sub)traces whose execution led to the PES and progression have been explored (all these sets are initialized as empty, see line 14). Finally, we add an empty (sub)trace ($\langle\rangle$) to the set of (sub)traces whose execution led to n_0 and progression needs to be explored (lines 17 and18), this empty (sub)trace will be the starting point of the automaton replay.

To explore the automaton behavior we pull the first element of the queue ($\widehat{n}$), we retrieve its set of (sub)traces to explore ($V_{\widehat{n}}$), and its set of outgoing arcs (i.e. all the arcs of the automaton (n_1, n_2, t) s.t. $n_1 = \widehat{n}$), see lines 20 to 22. For each (sub)trace $\widehat{\tau}$ whose execution led to $\widehat{n}$ and whose progression needs to be explored, we perform the following operations. We make a copy of $\widehat{\tau}$ (τ, line 24). If $\widehat{n}$ has no outgoing arcs ($O = \varnothing$), τ is either a full-trace (if its length is less than k, line 26) or a trace suffix, [9] accordingly, we add to E the edges (τ, s_0) and (s_0, τ) (this latter only in case of τ full-trace).

If $O \neq \varnothing$, for each outgoing arc $(\widehat{n}, n_t, t)$, we execute the following operations.

---

[9] I.e. no more activities can be executed because $\widehat{n}$ is a final PES, being $O = \varnothing$.

We update τ appending the activity t (line 32), this represents one of the possible progressions of the (sub)trace $\widehat{\tau}$. At this point three possible situations may occur: i) τ length is less than k, this means we did not observe enough activities to add τ as a state in S; ii) τ length is exactly k, this means τ is a prefix as it reached the length k after we appended the activity t, as a consequence we add τ to S and the edge (s_0, τ) to E (lines 33 to 35); iii) τ length is greater than k, and consequently we add $\tau^{2\rightarrow(k+1)}$ to S and $\left(\tau^{1\rightarrow k}, \tau^{2\rightarrow(k+1)}\right)$ to E (lines 37 to 39). Once S and E have been updated, we retrieve the set of (sub)traces explored from n_t (X_{n_t}), if the set does not contain $\tau^{2\rightarrow(k+1)}$, we add this latter to the set of (sub)traces to explore from n_t (V_{n_t}), and we add n_t to Q, if this latter does not already contain it (lines 40 to 44).

We repeat these steps for all the outgoing arcs of $\widehat{n}$, then, we update $V_{\widehat{n}}$ and $X_{\widehat{n}}$, moving the (sub)trace $\widehat{\tau}$ from $V_{\widehat{n}}$ to $X_{\widehat{n}}$ (lines 45 to 47). We iterate over these steps until Q is empty, after which M_P^k is completed.

Time Complexity. Algorithm 11 iterates on all the PES of the automaton, up to the number of incoming arcs of a PES (this is $2^{|\Omega|}$ in a behavioral automaton). At each iterations two nested loops are executed. The outer one, on the outgoing arcs of the PES, the inner one, on the (sub)traces to explore from the PES. Since in the worst case, each PES can have a number of outgoing arcs equal to the number of activities executable ($|\Omega|$), and the number of (sub)traces to explore from each PES is capped by the number of combinations of length k over the alphabet Ω, i.e. $k^{|\Omega|}$. The time complexity of Algorithm 11 is $O\left(2^{|\Omega|} \cdot |\Omega| \cdot k^{|\Omega|}\right) = O\left(|\Omega| \cdot 2k^{|\Omega|}\right)$.

6.3. Comparing Markovian Abstractions

In this section, we introduce our accuracy measures, the Markovian Abstraction-based fitness and precision. Both measures rely on the comparison of the process and the event log $\mathrm{M^k}$-abstractions. Given that the abstractions are graphs, it is possible to compare them applying either a structural comparator (e.g. a graph edit distance) or a simulation-based comparator. However, since the latter is computational expensive, we decided to opt for the former and to compare the $\mathrm{M^k}$-abstractions using a weighted edge-based graph matching algorithm.

Definition 6.3.1. ***[Weighted Edge-based Graph Matching Algorithm (GMA)]*** *A Weighted Edge-based Graph Matching Algorithm (GMA) is an algorithm that receives as input two graphs $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$, and outputs a mapping function $\mathscr{I}_C : E_1 \rightarrow (E_2 \cup \{\varepsilon\})$. The function $\mathscr{I}_C$ maps pairs of edges matched by the GMA or, if no mapping was found, the edges in E_1 are mapped to ε, i.e., $\forall e_1, e_2 \in E_1 : \mathscr{I}_C(e_1) = \mathscr{I}_C(e_2) \Rightarrow (e_1 = e_2) \vee (\mathscr{I}_C(e_1) = \varepsilon \wedge \mathscr{I}_C(e_2) = \varepsilon)$. A GMA is characterised by an underlying cost function $C : E_1 \times (E_2 \cup \{\varepsilon\}) \rightarrow [0,1]$, s.t. $\forall e_1 \in E_1$ and $\forall e_2 \in E_2 \Longrightarrow C(e_1, e_2) \in [0,1]$ and $\forall e_1 \in E_1 \Longrightarrow C(e_1, \varepsilon) =$*

Algorithm 11: Calculating M$^{\mathrm{k}}$-abstraction of a Process

```
input   : Process $P$
input   : Order $k$
Result: M$^{\mathrm{k}}$-abstraction $M_P^k$
Set $S \leftarrow \varnothing$;
Set $E \leftarrow \varnothing$;
State $s_0 \leftarrow -$;
add $s_0$ to $S$;
Graph $R \leftarrow generateAutomaton(P)$;
Set $N \leftarrow getPESs(R)$;
Node $n_0 \leftarrow getInititalPES(R)$;
Queue $Q \leftarrow \varnothing$;
add $n_0$ to $Q$;
Map $\mathscr{V} \leftarrow \varnothing$;
Map $\mathscr{X} \leftarrow \varnothing$;
for $n \in N$ do
    Set $V_n \leftarrow \varnothing$;
    Set $X_n \leftarrow \varnothing$;
    put $(n, V_n)$ in $\mathscr{V}$;
    put $(n, X_n)$ in $\mathscr{X}$;
Set $V_{n_0} \leftarrow getMapValue(\mathscr{V}, n_0)$;
add $\langle\rangle$ to $V_{n_0}$;
while $Q \neq \varnothing$ do
    $\widehat{n} \leftarrow poll(Q)$;
    $V_{\widehat{n}} \leftarrow getMapValue(\mathscr{V}, \widehat{n})$;
    Set $O \leftarrow getOutgoingArcs(R, \widehat{n})$;
    for $\widehat{\tau} \in V_{\widehat{n}}$ do
        Subtrace $\tau \leftarrow \widehat{\tau}$;
        if $O = \varnothing$ then
            if $|\tau| < k$ then
                add $\tau$ to $S$;
                add $(s_0, \tau)$ to $E$;
            add $(\tau, s_0)$ to $E$;
        else
            for $(\widehat{n}, n_t, t) \in O$ do
                $\tau \leftarrow \tau \oplus t$;
                if $|\tau| = k$ then
                    add $\tau$ to $S$;
                    add $(s_0, \tau)$ to $E$;
                else
                    if $|\tau| > k$ then
                        add $\tau^{2 \rightarrow (k+1)}$ to $S$;
                        add $\left(\tau^{1 \rightarrow k}, \tau^{2 \rightarrow (k+1)}\right)$ to $E$;
                $X_{n_t} \leftarrow getMapValue(\mathscr{X}, n_t)$;
                if $\tau^{2 \rightarrow (k+1)} \notin X_{n_t}$ then
                    $V_{n_t} \leftarrow getMapValue(\mathscr{V}, n_t)$;
                    add $\tau^{2 \rightarrow (k+1)}$ to $V_{n_t}$;
                    if $n_t \notin Q$ then add $n_t$ to $Q$ ;
        remove $\widehat{\tau}$ from $V_{\widehat{n}}$;
        $X_{\widehat{n}} \leftarrow getMapValue(\mathscr{X}, \widehat{n})$;
        put $\widehat{\tau}$ in $X_{\widehat{n}}$;
$M_P^k \leftarrow (S, E)$;
return $M_P^k$
```

1. *Hereinafter, we refer to any GMA as its mapping function* $\mathscr{I}_C$.

The GMA implemented in our measures is the Hungarian Algorithm [161, 162], which matches edges of E_1 to edges of E_2 to minimize the total cost of the matching, i.e. $\Sigma_{e_1 \in E_1} C(e_1, \mathscr{I}_C(e_1))$ is minimum. Furthermore, the time complexity of the Hungarian Algorithm is polynomial [163], excluding time complexity for computing the matching costs, i.e. $\forall e_1 \in E_1$ and $\forall e_2 \in E_2$, $C(e_1, e_2)$ is known. We note that, the underlying cost function plays a relevant role both in the time complexity and the accuracy of the comparison. We propose two alternative cost functions. The first is a boolean cost function, $C_b : E_1 \times (E_2 \cup \{\varepsilon\}) \to \{0,1\}$, s.t. $C_b(e_1, e_2) = 0 \iff e_1 = e_2$ otherwise $C_b(e_1, e_2) = 1$. The time complexity of C_b is constant: $O(1)$. The second cost function is the Levenshtein distance [164] normalised on $[0,1]$, we refer to it with the symbol C_l. We remind that in an M^{k}-abstraction each edge represents a (sub)trace of length $k+1$ (see Definition 6.2.1), therefore, we can compute the Levenshtein distance between the (sub)traces represented by two edges. The time complexity of C_l is $O\left(k^2\right)$ [165]. However, the difference between C_b and C_l is not only on their time complexity, the latter is more robust to noise, whilst the former is very strict.

In the following, we show how to compute the Markovian Abstraction-based fitness and precision, and the properties they fulfil. For the remaining part of the section, let L_x be a log, P_x be a process model, and $M^k_{L_x} = (S_{L_x}, E_{L_x})$ and $M^k_{P_x} = (S_{P_x}, E_{P_x})$ be the M^{k}-abstractions of the log and the model, respectively.

6.3.1. Markovian Abstraction-based Fitness

Given the GMA $\mathscr{I}_{C_b}$, an event log L and a process P as inputs, we compute the k^{th}-order Markovian abstraction-based fitness (hereby MAF^k) applying Equation 6.1.

$$MAF^k(L,P) = 1 - \frac{\Sigma_{e \in E_L} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e}{\Sigma_{e \in E_L} F_e} \tag{6.1}$$

Where F_e represents the frequency of the edge $e \in E_L$. Computing F_e is trivial while calculating the M^{k}-abstraction of the event log.[10] Furthermore, we decided to adopt C_b as underlying cost function for our Markovian-abstraction Fitness because choosing C_l would not guarantee the fulfilment of the Proposition-5 for fitness measures.

6.3.2. Proofs of the 7-Propositions of Fitness

In the following, we show that our Markovian abrastraction-based fitness measure fulfils the propositions presented in Section 6.1.

- **Proposition-1F.** $MAF^k(L,P)$ is a deterministic function. Given a log L and a process P, The construction of M^k_L and M^k_P is fully deterministic for $\mathscr{B}_P$ and $\mathscr{B}_L$ (see Definition 6.2.1). Furthermore, since the graph matching algorithm $\mathscr{I}_{C_b}$ is deterministic, as well as the $MAF^k(L,P)$ function of E_L, E_P

---

[10] This does not influence the time complexity of Algorithm 10.

and $\mathscr{I}_{C_b}$ (see Equation 6.1), it follows that $MAF^k(L,P)$ is also deterministic with codomain $[0,1]$ by definition.

- **Proposition-2F.** Given two process models P_1, P_2 and a log L, if $\mathscr{B}_{P_1} = \mathscr{B}_{P_2} \implies MAF^k(L,P_1) = MAF^k(L,P_2)$. From Theorem 6.2.1 the following relation holds: $E_{P_1} = E_{P_2}$. Since $MAF^k(L,P)$ is function of E_L, E_P and $\mathscr{I}_{C_b}$ (see Equation 6.1), it follows straightforward $MAF^k(L,P_1) = MAF^k(L,P_2)$.
- **Proposition-3.** Given two process models P_1, P_2 and a log L, if $\mathscr{B}_{P_1} \subseteq \mathscr{B}_{P_2} \implies MAF^k(L,P_2) \geq MAF^k(L,P_1)$. From Theorem 6.2.1 the following relation holds: $E_{P_1} \subseteq E_{P_2}$.

  Then, we distinguish two possible cases:
  1. if $E_{P_1} = E_{P_2}$, then $MAF^k(L,P_1) = MAF^k(L,P_2)$ follows straightforward (see Proposition-2 proof and Equation 6.1).
  2. if $E_{P_1} \subset E_{P_2}$, then the GMA would find matchings for either the same or a larger number of edges when applied on M^k_L and $M^k_{P_2}$ than when applied on M^k_L and $M^k_{P_1}$. Thus, a smaller or equal number of edges will be mapped to ε in the case of $MAF^k(L,P_2)$, not increasing the total matching cost $\Sigma_{e \in E_L} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e$, and guaranteeing $MAF^k(L,P_2) \geq MAF^k(L,P_1)$.
- **Proposition-4.** Given a process model P and two event logs L_1, L_2, if $\mathscr{B}_{L_2} \subseteq \mathscr{B}_P \implies MAF^k(L_1 \cup L_2, P) \geq MAF^k(L_1, P)$.

  Let $L_3 = L_1 \cup L_2$, $E_{L_3} = E_{L_2} \cup E_{L_1}$, and $E_d = E_{L_3} \setminus E_{L_1}$, it follows that if $e \in E_d \implies e \in E_{L_2} \setminus E_{L_1}$. Given that $\mathscr{B}_{L_2} \subseteq \mathscr{B}_P$, $\forall e_2 \in E_{L_2} \exists \widehat{e} \in E_P$ s.t. $e_2 = \widehat{e}$ (see Theorem 6.2.1). Consequently, the relation $E_d \subseteq E_P$ holds. Taking into account this latter result, we prove that $MAF^k(L_3,P) \geq MAF^k(L_1,P)$.

  $$1 - \frac{\Sigma_{e \in E_{L_3}} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e}{\Sigma_{e \in E_{L_3}} F_e} \geq 1 - \frac{\Sigma_{e \in E_{L_1}} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e}{\Sigma_{e \in E_{L_1}} F_e}$$

  We rewrite the relation.

  $$\frac{\Sigma_{e \in E_{L_1}} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e + \Sigma_{e \in E_d} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e}{\Sigma_{e \in E_{L_1} \cup E_d} F_e} \leq \frac{\Sigma_{e \in E_{L_1}} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e}{\Sigma_{e \in E_{L_1}} F_e}$$

  We note that, $\Sigma_{e \in E_d} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e = 0$, because $\forall e \in E_d \implies e \in E_{L_2}$ and $\exists \widehat{e} \in E_P$ s.t. $e = \widehat{e}$ and $C_b(e, \mathscr{I}_{C_b}(e)) = 0$. Removing the zero value from the relation, we obtain:

  $$\frac{\Sigma_{e \in E_{L_1}} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e}{\Sigma_{e \in E_{L_1} \cup E_d} F_e} \leq \frac{\Sigma_{e \in E_{L_1}} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e}{\Sigma_{e \in E_{L_1}} F_e}$$

This latter is always true, because $\Sigma_{e \in E_{L_1}} F_e < \Sigma_{e \in E_{L_1} \cup E_d} F_e$.

- **Proposition-5.** Given a process model P and two event logs L_1, L_2, if $\mathscr{B}_{L_2} \cap \mathscr{B}_P = \varnothing \implies MAF^k(L_1, P) \geq MAF^k(L_1 \cup L_2, P)$. Let $L_3 = L_1 \cup L_2$ and $E_d = E_{L_3} \setminus E_{L_1}$, two scenarios can materialise:
  1. $E_d \cap E_P = \varnothing$, this case occurs when none of the (sub)traces of length $k+1$ in $\mathscr{B}_{L_2}$ can be found among the (sub)traces in $\mathscr{B}_P$. It follows that $\forall e \in E_d$, $C_b(e, \mathscr{I}_{C_b}(e)) = 1$, because no matching edges can be found in the set E_P. Consequently, $\Sigma_{e \in E_{L_1} \cup\, E_d} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e$ will be greater than $\Sigma_{e \in E_{L_1}} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e$, and $MAF^k(L_1, P) > MAF^k(L_1 \cup L_2, P)$.
  2. $E_d \cap E_P \neq \varnothing$, this case occurs when there exists at least one (sub)trace of length $k+1$ in $\mathscr{B}_{L_2}$ that can be found among the (sub)traces in $\mathscr{B}_P$. in this case we cannot prove $MAF^k(L_1, P) \geq MAF^k(L_1 \cup L_2, P)$ for any k, but only for $k > k^*$, where k^* is the length of the longest (sub)trace in $\mathscr{B}_{L_2}$ that can be found among the (sub)traces in $\mathscr{B}_P$. Indeed, choosing $k > k^*$, we would fall in the first scenario. In the worst case, k^* is the length of the longest trace in L_2.
- **Proposition-6.** Given a process model P and an event log L, let $n \in \mathbb{N}$ and $L^n = \bigcup_n L \implies MAF^k(L, P) = MAF^k(L^n, P)$. The factor n alters the frequency of all the traces in L, but not its behavior, i.e. $\mathscr{B}_L = \mathscr{B}_{L^n}$. Given that the M^k-abstraction construction is only function of the behavior $M_L^k = M_{L^n}^k$, i.e. $S_L = S_{L^n}$ and $E_L = E_{L^n}$. However, MAF^k is function of the frequencies of the M^k-abstraction edges, and these frequencies will be affected by a factor n, s.t. $\forall e \in E_L$ and $\widehat{e} \in E_{L^n} \mid \widehat{e} = e \implies F_{\widehat{e}} = F_e \cdot n$. Nevertheless, n will not alter the value of MAF^k, as shown below.

$$MAF^k(L^n, P) = \frac{\Sigma_{e \in E_{L_1}} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e \cdot n}{\Sigma_{e \in E_L} F_e \cdot n} =$$

$$\frac{n \cdot \Sigma_{e \in E_{L_1}} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e}{n \cdot \Sigma_{e \in E_L} F_e} = MAF^k(L, P)$$

  It follows $MAF^k(L^n, P) = MAF^k(L, P)$.

- **Proposition-7.** Given a process model P and a log L, if $\mathscr{B}_L \subseteq \mathscr{B}_P \implies MAF^k(L, P) = 1$. From Theorem 6.2.1 the following relation holds: $E_L \subseteq E_P$, it follows that $\forall e \in E_L \implies C_b(e, \mathscr{I}_{C_b}(e)) = 0$, because for any e in E_L there exists an equal matching edge in E_P. Consequently, $\Sigma_{e \in E_L} C_b(e, \mathscr{I}_{C_b}(e)) \cdot F_e = 0$ and $MAF^k(L, P) = 1$.

6.3.3. Markovian Abstraction-based Precision

Given the GMA $\mathscr{I}_{C_l}$, an event log L and a process P as inputs, we can compute the k$^{\text{th}}$-order Markovian abstraction-based precision applying Equation 6.2.

	Propositions and Axioms									
	1P = A1	**2P = A4**	**8 ⊃ A2**	**9 = A5**	**10**	**11**	**12**	**13**	**A2**	**A3**
MAP^k w/ C_b	$\surd$ $(\forall k)$		$\surd$ $(\hat{k})$	$\surd$ $(\forall k)$	$\surd$ $(\hat{k})$	$\surd$ $(\forall k)$				$\surd$ (k^*)
MAP^k w/ C_l	$\surd$ $(\forall k)$		×	$\surd$ $(\forall k)$	$\surd$ $(\hat{k})$	$\surd$ $(\forall k)$				$\surd$ (k^*)

Table 29: Precision propositions [4] and axioms [5] fulfiled by MAP^k, and for which value of k.

$$MAP^k_{\mathscr{I}_{C_l}}(L,P) = 1 - \frac{\Sigma_{e \in E_P} C_l(e, \mathscr{I}_{C_l}(e))}{|E_P|} \tag{6.2}$$

However, while this measure can fulfil all the precision Axioms proposed in [5], as we showed in a previous study [19], it cannot fulfil Proposition-8. Therefore, similarly to the case of MAF^k, we need to set the GMA to $\mathscr{I}_{C_b}$ and define the k$^{\text{th}}$-order Markovian abstraction-based precision as Equation 6.3.

$$MAP^k(L,P) = 1 - \frac{\Sigma_{e \in E_P} C_b(e, \mathscr{I}_{C_b}(e))}{|E_P|} \tag{6.3}$$

Recollecting the BPMN models shown in Figure 22 and their respective Markovian abstractions (for $k = 1$ and $k = 2$, Figures 23a-23c and 24a-24c). We can observe that by increasing k the quality of the behavioral abstractions increases, capturing more details. Consequently, our MAP^k outputs a finer result. Note that, our proposed precision measure fulfils the properties of an ordinal scale. Specifically, given an event log $\mathscr{L}$ and a k, MAP^k induces an order over the possible process models that fit log $\mathscr{L}$. This property is desirable given that the purpose of a precision measure is to allow us to compare two different process models in terms of their extra behavior.

6.3.4. Proofs of the 8-Propositions of Precision

In the following, we show that our Markovian abrastraction-based precision measure fulfils the propositions presented in Section 6.1, as well as Axiom-3.

- **Proposition-1P.** $MAP^k(L,P)$ is a deterministic function. Given a log L and a process P, The construction of M^k_L and M^k_P is fully deterministic for $\mathscr{B}_P$ and $\mathscr{B}_L$ (see Definition 6.2.1). Furthermore, since the graph matching algorithm $\mathscr{I}_{C_b}$ is deterministic, as well as the $MAP^k(L,P)$ function of E_L, E_P and $\mathscr{I}_{C_b}$ (see Equation 6.3), it follows that $MAP^k(L,P)$ is also deterministic with codomain $[0,1]$ by definition.
- **Proposition-2P.** Given two processes P_1, P_2 and an event log L, s.t. $\mathscr{B}_{P_1} = \mathscr{B}_{P_2}$ then $MAP^k(L,P_1) = MAP^k(L,P_2)$. If $\mathscr{B}_{P_1} = \mathscr{B}_{P_2}$, then $E_{P_1} = E_{P_2}$ (see Theorem 6.2.1). It follows straightforward that $MAP^k(L,P_1) = MAP^k(L,P_2)$ (see proof Proposition-1P and Equation 6.3).
- **Proposition-8.** Given two processes P_1, P_2 and a log L, if $\mathscr{B}_{P_1} \subseteq \mathscr{B}_{P_2}$ and $\mathscr{B}_L \cap (\mathscr{B}_{P_2} \setminus \mathscr{B}_{P_1}) = \varnothing$, then $MAP^k(L,P_1) \geq MAP^k(L,P_2)$. The relation $E_{P_1} \subseteq E_{P_2}$ holds for Theorem 6.2.1, and we distinguish two possible cases:

1. $E_{P_1} = E_{P_2}$. In this case, it follows straightforward $MAP^k(L,P_1) = MAP^k(L,P_2)$ (see Proposition-1P proof and Equation 6.3).
2. $E_{P_1} \subset E_{P_2}$. Let $E_{P_2} = E_{P_1} \cup E_x$, then:

$$MAP^k(L,P_1) - MAP^k(L,P_2) =$$

$$1 - \frac{\Sigma_{e \in E_{P_1}} C_b(e, \mathscr{I}_{C_b}(e))}{|E_{P_1}|} - 1 + \frac{\Sigma_{e \in E_{P_2}} C_b(e, \mathscr{I}_{C_b}(e))}{|E_{P_2}|} =$$

$$\frac{\Sigma_{e \in E_{P_1}} C_b(e, \mathscr{I}_{C_b}(e)) + \Sigma_{e \in E_x} C_b(e, \mathscr{I}_{C_b}(e))}{|E_{P_1}| + |E_x|} - \frac{\Sigma_{e \in E_{P_1}} C_b(e, \mathscr{I}_{C_b}(e))}{|E_{P_1}|}$$

   We prove that $MAP^k(L,P_1) - MAP^k(L,P_2) \geq 0$ by negating it.

$$\frac{\Sigma_{e \in E_{P_1}} C_b(e, \mathscr{I}_{C_b}(e)) + \Sigma_{e \in E_x} C_b(e, \mathscr{I}_{C_b}(e))}{|E_{P_1}| + |E_x|} - \frac{\Sigma_{e \in E_{P_1}} C_b(e, \mathscr{I}_{C_b}(e))}{|E_{P_1}|} < 0$$

$$\left(1 - \frac{|E_{P_1}| + |E_x|}{|E_{P_1}|}\right) \Sigma_{e \in E_{P_1}} C_b(e, \mathscr{I}_{C_b}(e)) + \Sigma_{e \in E_x} C_b(e, \mathscr{I}_{C_b}(e)) < 0$$

$$\Sigma_{e \in E_x} C_b(e, \mathscr{I}_{C_b}(e)) - \frac{|E_x|}{|E_{P_1}|} \Sigma_{e \in E_{P_1}} C_b(e, \mathscr{I}_{C_b}(e)) < 0 \qquad (6.4)$$

   By definition, the following two relations always hold:

$$0 \leq \Sigma_{e \in E_x} C_b(e, \mathscr{I}_{C_b}(e)) \leq |E_x|$$

$$0 \leq \Sigma_{e \in E_{P_1}} C_b(e, \mathscr{I}_{C_b}(e)) \leq |E_{P_1}|$$

   Relation 6.4 is always false when $\Sigma_{e \in E_x} C_b(e, \mathscr{I}_{C_b}(e)) = |E_x|$, i.e. when none of the edges in E_x can be found in E_L ($E_x \cap E_L = \varnothing$). In general, the latter does not hold for any k, because E_x captures all the (sub)traces of $\mathscr{B}_{P_2} \setminus \mathscr{B}_{P_1}$, some of which could be found in $\mathscr{B}_L$. However, similarly to the case 2 of Proposition-5, to fulfil $E_x \cap E_L = \varnothing$ (and consequently Proposition-8), it is sufficient to set $k > k^*$. Where k^* is the length of the longest (sub)trace in $\mathscr{B}_L$ that can be found among the (sub)traces in $\mathscr{B}_{P_2} \setminus \mathscr{B}_{P_1}$. In the worst scenario, k^* is the length of the longest trace in L.

- **Proposition-9.** Given two event logs L_1, L_2 and a process P, s.t. $\mathscr{B}_{L_1} \subseteq \mathscr{B}_{L_2} \subseteq \mathscr{B}_P$, then $MAP^k(L_2,P) \geq MAP^k(L_1,P)$. Consider the two following cases:
   1. if $\mathscr{B}_{L_1} = \mathscr{B}_{L_2}$, then $E_{L_1} = E_{L_2}$ (see Theorem 6.2.1).
      It follows $MAP^k(L_2,P) = MAP^k(L_1,P)$, because $MAP^k(L,P)$ is a deterministic function of E_L, E_P and $\mathscr{I}_{C_b}$ (see Proposition-1P proof and Equation 6.3).

2. if $\mathscr{B}_{L_1} \subset \mathscr{B}_{L_2}$, then $E_{L_1} \subseteq E_{L_2}$ (see Theorem 6.2.1). In this case, the graph matching algorithm would find matchings for either the same number or a larger number of edges between M_P^k and $M_{L_2}^k$, than between M_P^k and $M_{L_1}^k$ (this follows from $E_{L_1} \subseteq E_{L_2}$). Thus, a smaller or equal number of edges will be mapped to ε in the case of $MAP^k(L_2,P)$ not decreasing the value for the precision, i.e., $MAP^k(L_2,P) \geq MAP^k(L_1,P)$.

- **Proposition-10.** Given a process model P and two event logs L_1, L_2, s.t. $\mathscr{B}_{L_1} \subseteq \mathscr{B}_{L_2}$ and $\mathscr{B}_P \cap (\mathscr{B}_{L_2} \setminus \mathscr{B}_{L_1}) = \varnothing$, then $MAP^k(L_1,P) = MAP^k(L_2,P)$. Let us consider the following two sets: $E_i = E_P \cap E_{L_2}$ and $E_o = E_P \setminus E_{L_2}$, we can compute $MAP^k(L_2,P)$ as follows:

$$MAP^k(L_2,P) = 1 - \frac{\Sigma_{e \in E_P} C_b(e, \mathscr{I}_{C_b}(e))}{|E_P|} =$$

$$1 - \frac{\Sigma_{e \in E_i} C_b(e, \mathscr{I}_{C_b}(e)) + \Sigma_{e \in E_o} C_b(e, \mathscr{I}_{C_b}(e))}{|E_P|} = 1 - \frac{|E_o|}{|E_P|}$$

$\Sigma_{e \in E_i} C_b(e, \mathscr{I}_{C_b}(e)) = 0$
holds by construction, because $\forall e \in E_i \implies C_b(e, \mathscr{I}_{C_b}(e)) = 0$, since for each edge in E_i there exists an equal edge in E_{L_2}. The equality $\Sigma_{e \in E_o} C_b(e, \mathscr{I}_{C_b}(e)) = |E_o|$ holds by construction because $\forall e \in E_o \implies C_b(e, \mathscr{I}_{C_b}(e)) = 1$, since none of the edges in E_o can be matched to any edge in E_{L_2}. Similarly, if we consider $E_x = E_P \cap E_{L_1}$ and $E_y = E_P \setminus E_{L_1}$, we can compute $MAP^k(L_1,P) = 1 - \frac{|E_y|}{|E_P|}$
Let $E_d = E_{L_2} \setminus E_{L_1}$, and $E_{L_2} = E_{L_1} \cup E_d$, two scenarios can materialise:

  1. $E_P \cap E_d = \varnothing$.
     In this case, $E_o = E_P \setminus (E_{L_1} \cup E_d) = E_P \setminus E_{L_1} = E_y$. It follows that $MAP^k(L_1,P) = MAP^k(L_2,P)$.
  2. $E_P \cap E_d \neq \varnothing$, this case occurs when there exists at least one (sub)trace of length $k+1$ in $\mathscr{B}_P$ that can be found among the (sub)traces in $\mathscr{B}_{L_2} \setminus \mathscr{B}_{L_1}$. In this case we cannot prove $MAP^k(L_1,P) = MAP^k(L_2,P)$ for any k, but only for $k > k^*$, where k^* is the length of the longest (sub)trace in $\mathscr{B}_{L_2}$ that can be found among the (sub)traces in $\mathscr{B}_{L_2} \setminus \mathscr{B}_{L_1}$. Indeed, choosing $k > k^*$, we would fall in the first scenario. In the worst case, k^* is the length of the longest trace in $L_2 \setminus L_1$.

- **Proposition-11.** Given a process model P and an event log L, let $n \in \mathbb{N}$ and $L^n = \bigcup_n L$, then $MAP^k(L,P) = MAP^k(L^n,P)$. This proposition holds straightforward since $\mathscr{B}_L = \mathscr{B}_{L^n}$ and $E_L = E_{L^n}$ for Theorem 6.2.1, while $MAP^k(L,P)$ is a deterministic function of E_L, E_P, and $\mathscr{I}_{C_b}$.

- **Proposition-12.** Given a process model P and a log L, if $\mathscr{B}_P = \mathscr{B}_L$, then $MAP^k(L,P) = 1$. The relation $E_P = E_L$ holds for Theorem 6.2.1, it follows that $\forall e \in E_P \Longrightarrow C_b(e, \mathscr{I}_{C_b}(e)) = 0$, since for each edge in E_P there exists an equal edge in E_L. Then, by definition, $MAP^k(L,P) = 1$ (see Equation 6.3).

- ○ **Proposition-13.** Given a process model P and a log L, if $\mathscr{B}_P \subseteq \mathscr{B}_L$, then $MAP^k(L,P) = 1$. If $\mathscr{B}_P \subseteq \mathscr{B}_L$, then $E_P \subseteq E_L$ (see Theorem 6.2.1). As in Proposition-12, $\forall e \in E_P \Longrightarrow C_b(e, \mathscr{I}_{C_b}(e)) = 0$, since for each edge in E_P there exists an equal edge in E_L. It follows that $MAP^k(L,P) = 1$ (see Equation 6.3).
- ○ **Axiom-3.** Given two processes P_1, P_2 and an event log L, s.t. $\mathscr{B}_L \subseteq \mathscr{B}_{P_1} \subset \mathscr{B}_{P_2} = \Gamma_\Omega$ then $MAP^k(L,P_1) > MAP^k(L,P_2)$. The relation $MAP^k(L,P_1) \geq MAP^k(L,P_2)$ holds for Proposition-8. The case $MAP^k(L,P_1) = MAP^k(L,P_2)$ occurs when $M^k_{P_2}$ over-approximates the behavior of P_2, i.e. $\mathscr{B}_{P_1} \subset \mathscr{B}_{P_2}$ and $E_{P_1} = E_{P_2}$. Nevertheless, for any $\mathscr{B}_{P_1}$ there always exists a k^* s.t. $E_{P_1} \subset E_{P_2}$. This is true since $\mathscr{B}_{P_1}$ is strictly contained in $\mathscr{B}_{P_2}$, there exists a trace $\widehat{\tau} \in \mathscr{B}_{P_2}$ s.t. $\widehat{\tau} \notin \mathscr{B}_{P_1}$. Choosing $k^* = |\widehat{\tau}|$, the $M^{k^*}_{P_2}$ would produce an edge $\widehat{e} = (-, \widehat{\tau}) \in E_{P_2}$ s.t. $\widehat{e} \notin E_{P_1}$ because $\widehat{\tau} \notin \mathscr{B}_{P_1}$ (see also Definition 6.2.1).[11] Consequently, for any $k \geq k^*$, we have $E_{P_1} \subset E_{P_2}$ and $MAP^k(L,P_1) > MAP^k(L,P_2)$ holds, since this latter is the case 2 of Axiom-2.

In Axiom-3 we showed that there exists a specific value of k, namely k^*, for which $MAP^{k^*}(L_x, P_x)$ satisfies Axiom-3 and we identified such value as $k^* = |\widehat{\tau}|$, where $\widehat{\tau}$ can be any trace of the set difference $\Gamma_\Omega \setminus \mathscr{B}_{P_x}$. In the following, we show how to identify the minimum value of k^*. To identify the lowest value of k^*, we have to consider the traces $\widehat{\tau} \in \Gamma_\Omega$ such that does not exists a $\tau \in \mathscr{B}_{P_x}$ where $\widehat{\tau} \subseteq \tau$. If a trace $\widehat{\tau} \in \Gamma_\Omega$ that is not a sub-trace of any other trace of the process model behavior ($\mathscr{B}_{P_x}$) is found, by setting $k^* = |\widehat{\tau}|$ would mean that in the M$^{k^*}$-abstraction of Γ_Ω there will be a state $\widehat{s} = \widehat{\tau}$ and an edge $(-, \widehat{\tau})$ that are not captured by the M$^{k^*}$-abstraction of $\mathscr{B}_{P_x}$. This difference will allow us to distinguish the process P_x from the flower model (i.e. the model having a behavior equal to Γ_Ω), satisfying in this way the Axiom-3. At this point, considering the set of the lengths of all the subtraces not contained in any trace of $\mathscr{B}_{P_x}$, $Z = \{|\widehat{\tau}| \,:\, \widehat{\tau} \in \Gamma_\Omega \wedge \nexists\, \tau \in \mathscr{B}_{P_x} \mid \widehat{\tau} \subseteq \tau\}$, we can set the lower-bound of $k^* \geq min(Z)$.

Note that the value of k^* is equal to 2 for any process model with at least one activity that cannot be executed twice in a row. If we have an activity $\widehat{t}$ that cannot be executed twice in a row, it means that $\left|\langle \widehat{t}, \widehat{t} \rangle\right| \in Z$ and thus we can set $k^* = 2$. In practice, $k^* = 2$ satisfies Axiom-3 in real-life cases, since it is very common to find process models that have the above topological characteristic.

Table 29 summarises the propositions and axioms fulfilled by MAP^k when adopting a GMA with C_b or C_l and for which value of k, where $\hat{k}$ represents the length of the longest trace in the event log, and k^* the lower bound of the set Z (as discussed above).

---

[11] Formally, $\exists \widehat{\tau} \in \mathscr{B}_{P_2} \setminus \mathscr{B}_{P_1}$, s.t. for $k^* = |\widehat{\tau}| \Longrightarrow \exists (-, \widehat{\tau}) \in E_{P_2} \setminus E_{P_1}$.

6.4. Evaluation

In this section, we report on a two-pronged evaluation we performed in order to: (i) compare the fitness and precision results yielded by our measures to those yielded by the state-of-the-art measures; (ii) assess to what extend our measures outperform the time performance of the state-of-the-art measures; and (iii) analyse the role of the order k. We set up a qualitative evaluation based on artificial data to understand whether our measures yield results that are in-line with or more intuitive than those of the state-of-the-art measures. At the same time, the qualitative evaluation allow us to comprehend the value of the fitness and precision measures properties (introduced in Section 6.1). On the other hand, for assessing the time performance of our measures, we set up a quantitative evaluation based on real-life data. This latter evaluation is the most crucial in our context, given that our main goal is the designing of scalable accuracy measures.

To do so, we implemented the Markovian Abstraction-based Fitness and Precision (MAF^k and MAP^k) in a standalone open-source tool[12] and used it to carry out a qualitative evaluation on synthetic data and a quantitative evaluation on real-life data.[13]

All the experiments were executed on an Intel Core i5-6200U @2.30GHz with 16GB RAM running Windows 10 Pro (64-bit) and JVM 8 with 12GB RAM (8GB Stack and 4GB Heap). For each measurement we set a timeout of two hours.

6.4.1. Qualitative Evaluation Dataset

In a previous study, van Dongen et al. [2] showed that their anti-alignment precision was able to generate more intuitive rankings of model-log pairs than existing state-of-the-art precision measures using a synthetic dataset of model-log pairs. Table 30 shows the synthetic event log used in [2], while Figure 25a shows the "original model" that was used in that paper to derive eight process model variants. The set of models includes a *single trace* model capturing the most frequent trace, Fig. 25b; a model incorporating all *separate traces*, Fig. 25c; a *flower model* of all activities in the log, Fig. 25d; a model with activities G and H in parallel (*Opt. G || Opt. H*, see Fig. 25e); one with G and H in self-loop (*↺G, ↺H*, Fig. 25f); a model with D in self-loop (*↺D*, Fig. 25g); a model with all activities in parallel (*All parallel*, Fig. 25h); and a model where all activities are in round robin (*Round robin*, Fig. 25i).

To evaluate the MAF^k fitness measures, we generated from each of the above models, an event log containing the whole behavior of the process model. In doing so, we capped the number of times each cycle in the model was executed to two iterations each. We held out the *Flower* model because we were unable to generate its corresponding event log, given that this model has almost one million

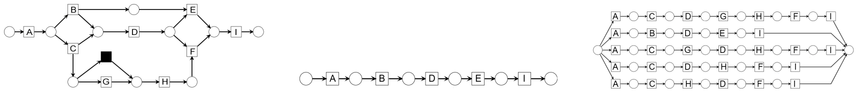
---

[12] Available at `http://apromore.org/platform/tools`

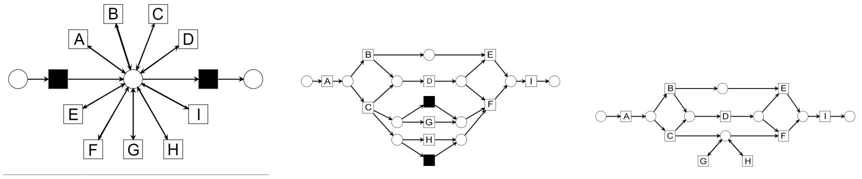
[13] The public data used in the experiments can be found at `http://doi.org/10.6084/M9.FIGSHARE.7397006.V1`

Traces	#
$\langle A,B,D,E,I\rangle$	1207
$\langle A,C,D,G,H,F,I\rangle$	145
$\langle A,C,G,D,H,F,I\rangle$	56
$\langle A,C,H,D,F,I\rangle$	23
$\langle A,C,D,H,F,I\rangle$	28

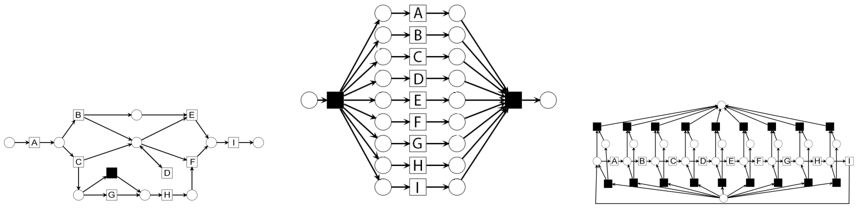
Table 30: Test log [2].



(a) Original model.   (b) Single trace.   (c) Separate traces.



(d) Flower model.   (e) Opt. G || Opt. H model.   (f) ↺G, ↺H model.



(g) ↺D model.   (h) All parallel model.   (i) Round robin model.

Figure 25: Qualitative evaluation artificial models as in [2].

unique traces if each cycle is executed at most twice. This led to a total of eight model-log pairs.

Below, we evaluate the proposed fitness and precision measures using the above model-log pairs.

6.4.2. Qualitative Evaluation of MAF^k

Using the original model (Figure 25a), we measured the fitness of the eight logs synthetized from the process model variants. We compared our fitness measure MAF^k to the fitness measures discussed in Section 6.1, namely: token-based fitness (*Token*), alignment-based fitness (*Alignment*), PCC fitness, with projections size equal to 10 (PCC_{10}), [14] and entropy-based fitness both exact [146] (*EBF*) and partial matching [166] (EBF_p) variants.

Table 31 gives the numerical results of the fitness measurements. We note that some of the fitness values returned by the *Token* measure are counter-intuitive. For example, for the log generated from the original model, this measure scores a value different than 1 while this model fully fits this log. As such, this result brakes Proposition-7 of fitness. Other counter-intuitive values are returned for the logs generated from the *separate traces* and the *all parallel* models, respectively 0.951 and 0.556. The former should also be 1 as per Proposition-7, whilst the latter, given that none of the log traces are contained in the original model behavior, it should return a value close to 0.

Alignment, PCC_{10}, and EBF_p also return high values of fitness for the *all parallel* log, respectively 0.464, 1.0, and 0.658. These results are also counter-intuitive if we consider that the *all parallel* log contains more than 350 thousand different traces and none of them is contained in the original model behavior. A similar reasoning can be done for the fitness values of *Token*, *Alignment*, and EBF_p on the log generated from the *round robin* model. This latter log contains no traces of the original process behavior, yet the fitness values yielded by the three measures are very high (respectively 0.655, 0.501, and 0.638). These results, at a glance counter-intuitive, can be attributed to the fact that the three fitness measures allow for partial matching, i.e. they do not penalise traces in the event log that are similar (up to a certain degree) to one or more traces in the process behavior. On the other hand, *EBF*, not allowing for partial matching, returns a fitness value of 0 for both the *all parallel* and the *round robin* logs.

As for our fitness measure MAF^k, we note that the numeric values for all the logs generated from the acyclic models stabilize at k equal to 4, whilst the values for the logs generated from the three cyclic models (1: ↺*G*, ↺*H*, 2: ↺*D* and 3: *Round robin*) tend to 0 as we increase k. This is expected as the higher the k, the larger the behavior captured in the M^{k}-abstractions, and consequently the more precise the measurement. In line with Proposition-7, MAF^k returns a fitness value

---

[14] We chose a size of 10 for PCC because this was the highest value for which we were able to obtain more than 50% of measurements.

Log	*Token*	*Align.*	PCC_{10}	*EBF*	EBF_p
Original model	0.960	1.000	1.000	1.000	1.000
Single trace	1.000	1.000	1.000	1.000	1.000
Separate traces	0.951	1.000	1.000	1.000	1.000
Opt. G || Opt. H	0.920	0.956	0.500	0.908	0.987
↻G, ↻H	0.880	0.918	0.231	0.853	0.958
↻D	0.781	0.873	0.171	0.839	0.978
All parallel	0.556	0.464	1.000	0.000	0.658
Round robin	0.655	0.501	0.000	0.000	0.638

Log	MAF^1	MAF^2	MAF^3	MAF^4	MAF^5	MAF^6	MAF^7
Original model	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Single trace	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Separate traces	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Opt. G || Opt. H	0.889	0.679	0.563	0.500	0.500	0.500	0.500
↻G, ↻H	0.889	0.633	0.409	0.259	0.167	0.143	0.200
↻D	0.889	0.633	0.419	0.250	0.138	0.094	0.109
All parallel	0.222	0.038	0.006	0.001	< 0.001	< 0.001	0.000
Round robin	0.444	0.000	0.000	0.000	0.000	0.000	0.000

Table 31: Values of the fitness measures over the synthetic dataset.

of 1 for all the logs fully fitting the original model (first three rows of Table 31). In contrast with the state of the art, MAF^k is able to identify the large difference between the behavior of the original model and that recorded in the *all parallel* log, as it returns a value close to 0 already for k equal to 2, similarly to the *round robin* log. The only measures that return a value close to 0 for the *round robin* log are PCC_{10} and *EBF*, besides our measure.

Since each fitness measure assesses the fitness of a log over a model in a different way, it is useful to analyse the ranking yielded by each measure for the set of logs in question. This is shown in Table 32, where we report also a reference ranking generated by applying the strict definition of fitness given in Section 3.1. The reference ranking is derived by ordering the fitness values of each pair log-model computed as the ratio of the number of traces in the log that belong to the model behaviour over the number of traces in the log. We note that the rankings yielded by *Token* and PCC_{10} differ the most from the reference one. *Alignment* and MAF^k (with k between 4 and 6) assign the highest rank to the fully fitting logs, followed by the partially fitting logs with acyclic behavior (i.e. *Opt. G* || *Opt. H*), then by the logs containing cyclic behavior (*↻G*, *↻H*, and *↻D*) and finally by the two completely unfitting logs (*all parallel* and *round robin*). Except for the *all parallel* and *round robin* logs, the rankings yielded by *Alignment* and MAF^k (with k between 4 and 6) match the reference ranking. Lastly, *EBF* and MAF^7] agree with the reference ranking, highlighting that the two measures operationalize the strict definition of fitness.

6.4.3. Qualitative Evaluation of MAP^k

Using the original log (as in Table 30), we measured the precision of the nine model variants (shown in Fig. 25a-25i), and compared our precision measure MAP^k (with C_l) against the baseline measures discussed in Section 6.1, specifi-

Log	*Reference*	*Token*	*Align.*	PCC_{10}	*EBF*	EBF_p
Original model	6	7	6	5	6	6
Single trace	6	8	6	5	6	6
Separate traces	6	6	6	5	6	6
Opt. G || Opt. H	5	5	5	4	5	5
↺G, ↺H	4	4	4	3	4	3
↺D	3	3	3	2	3	4
All parallel	1	1	1	5	1	2
Round robin	1	2	2	1	1	1

Log	*Reference*	MAP^1	MAF^2	MAF^3	MAF^4	MAF^5	MAF^6	MAF^7
Original model	6	6	6	6	6	6	6	6
Single trace	6	6	6	6	6	6	6	6
Separate traces	6	6	6	6	6	6	6	6
Opt. G || Opt. H	5	2	5	5	5	5	5	5
↺G, ↺H	4	2	3	3	4	4	4	4
↺D	3	2	3	4	3	3	3	3
All parallel	1	1	2	2	2	2	2	1
Round robin	1	5	1	1	1	1	1	1

Table 32: Model ranking induced by the fitness measures over the synthetic dataset.

cally trace-set difference precision (*SD*), alignment-based ETC precision (ETC_a), negative events precision (*NE*), projected conformance checking with its projections size equal to 2 (PCC_2), [15] anti-alignment precision (*AA*), and entropy-based precision both exact [146] (*EBP*) and partial matching [166] (EBP_p) variants. We left out advanced behavioral appropriateness (*ABA*) as it is undefined for some of the models in this dataset. We limited the order k to 7, because it is the length of the longest trace in the log. Setting an order greater than 7 would only (further) penalise the cyclic behavior of the models.

Table 33 reports the results of our qualitative evaluation.To discuss these results, we use three precision measures as reference. The first one is *SD*, as it closely operationalizes the definition of precision (as given in Section 3.1) by capturing the exact percentage of model behavior that cannot be found in the log. Though, as discussed in Section 6.1, *SD* can only be computed for acyclic models, and uses by design a value of zero for cyclic models. To have a reference for cyclic models, we considered *AA* and EBP_p. The former (*AA*) because it has been shown [2] to be intuitively more accurate than other precision measures. The latter (EBP_p), the most recent precision measure, because it has been shown to be mostly in line with *AA* [166], while also fulfilling the precision propositions.

From the results in Table 33, we can observe that MAP^1 does not penalise enough the extra behavior of some models, such as the *original* model, which cannot be distinguished from the *single trace* and the *separate traces* models (all have a precision of 1). Also, the values of MAP^1 are far from those of *SD* (with the exception of the simplest models, i.e. *single trace* and *separate traces*), yet close to those of both *AA* and EBP_p. As we increase k, MAP^k tends to *SD*. We note that,

---

[15] We chose size 2 for PCC since this is the highest value for which we were able to obtain more than 50% of the measurements.

Process Variant	*Traces (#)*	*SD*	ETC_a	*NE*	PCC_2	*AA*	*EBP*	EBP_p
Original model	6	0.833	0.900	0.995	1.000	0.871	0.979	0.998
Single trace	1	1.000	1.000	0.893	1.000	1.000	1.000	1.000
Separate traces	5	1.000	1.000	0.985	1.000	1.000	1.000	1.000
Flower model	986,410	0.000	0.153	0.117	0.509	0.000	0.125	0.479
Opt. G || Opt. H	12	0.417	0.682	0.950	0.991	0.800	0.889	0.986
↺G, ↺H	362	0.000	0.719	0.874	0.889	0.588	0.568	0.933
↺D	118	0.000	0.738	0.720	0.937	0.523	0.758	0.970
All parallel	362,880	0.000	0.289	0.158	0.591	0.033	0.000	0.656
Round robin	27	0.000	0.579	0.194	1.000	0.000	0.000	0.479

Process Variant	*Traces (#)*	MAP^1	MAP^2	MAP^3	MAP^4	MAP^5	MAP^6	MAP^7
Original model	6	1.000	0.895	0.833	0.786	0.778	0.833	0.833
Single trace	1	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Separate traces	5	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Flower model	986,410	0.176	0.021	0.002	t/o	t/o	t/o	t/o
Opt. G || Opt. H	12	0.889	0.607	0.469	0.393	0.389	0.417	0.417
↺G, ↺H	362	0.800	0.370	0.134	0.041	0.011	0.003	0.001
↺D	118	0.889	0.515	0.273	0.128	0.055	0.028	0.021
All parallel	362,880	0.222	0.034	0.005	0.005	t/o	t/o	t/o
Round robin	27	0.600	0.467	0.425	0.340	0.267	0.200	0.213

Table 33: Values of the precision measures over the synthetic dataset.

Process Variant	*SD*	ETC_a	*NE*	PCC_2	*AA*	*EBP*	EBP_p
Original model	7	7	9	6	7	7	7
Single trace	8	8	6	6	8	8	8
Separate traces	8	8	8	6	8	8	8
Flower model	1	1	1	1	1	3	1
Opt. G || Opt. H	6	3	7	5	6	6	6
↺G, ↺H	1	5	5	3	5	4	4
↺D	1	6	4	4	4	5	5
All parallel	1	2	2	2	3	1	3
Round robin	1	4	3	6	1	1	1

Process Variant	MAP^1	MAP^2	MAP^3	MAP^4	MAP^5	MAP^6	MAP^7
Original model	7	7	7	7	7	7	7
Single trace	7	8	8	8	8	8	8
Separate traces	7	8	8	8	8	8	8
Flower model	1	1	1	1	1	1	1
Opt. G || Opt. H	5	6	6	6	6	6	6
↺G, ↺H	4	3	3	3	3	3	3
↺D	5	5	4	4	4	4	4
All parallel	2	2	2	2	2	2	2
Round robin	3	4	5	5	5	5	5

Table 34: Model rankings induced by the precision measures over the synthetic dataset.

the main difference between the values of MAP^k, *SD*, *AA*, and EBP_p is captured by the manner these measures penalise the extra cyclic behavior allowed by a model. MAP^k tends to zero quickly as we increase k, *SD* harshly penalise cyclic models returning precision values of zero, while *AA* and EBP_p do not penalise beyond a certain level, with EBP_p being the most forgiven. In this respect, let us consider the cyclic models in our datasets: i) the *flower* model, ii) the ↺*G*, ↺*H* model (Fig. 25f), iii) the ↺*D* model (Fig. 25g), and iv) the *round robin* (Fig. 25i). The value of our precision measure immediately drops to zero in the *flower* model ($k = 3$) because this latter model allows the greatest amount of cyclic behavior, due to all the possible combinations of activities being permitted. This is consistent with both *SD* and *AA*, while EBP_p returns the highest value of precision among the reference measures (precisely, 0.479).

On the contrary, MAP^k tends to zero slower in the *round robin* model because this model is very strict on the order in which activities can be executed, despite having infinite behavior. In fact, it only allows the sequence $\langle A, B, C, D, F, G, H, I \rangle$ to be executed, with the starting activity and the number of repetitions being variable. This is taken into account by our measure, since even with $k = 7$ we do not reach a value of zero for this model, as opposed to *SD* and *AA*. This allows us to discriminate the *round robin* model from other models with very large behavior such as the *flower* model. This is not possible with *SD*, *AA*, and EBP_p, indeed *SD* and *AA* returns a precision of zero for both models, while in contrast EBP_p returns a precision of 0.479 for both models.

As for the other two cyclic models in our dataset, MAP^k tends to zero with speeds between those of the *flower* model and the *round robin* model, with the ↺*G*, ↺*H* model dropping faster than the ↺*D*, due to the former allowing more cyclic behavior than the latter. Similar considerations as above apply to these two models. Even for $k = 7$, their precision does not reach zero, allowing us to distinguish the precision of these two models. Instead, *SD* sets the precision of the two models to zero by design. If we consider *AA*, we note that it assigns precision values well above zero to the two models, with the ↺*G*, ↺*H* model having a higher precision than the ↺*D* model (0.588 vs. 0.523). This result is counter-intuitive, since the former model allows more behavior not permitted by the log (in terms of number of different traces) than the latter model does. In addition, *AA* penalizes more the *round robin* model, despite this latter has less behavior than the two models with self-loop activities.

Similarly to *AA* also EBP_p is generous in assessing the precision of the ↺*G*, ↺*H* model and the ↺*D* model, setting their precision at 0.933 and 0.970 (respectively), very close to the original model precision. As for the case of EBF_p, this is due to the partial matching nature of EBP_p. Indeed, we note that its exact variant (*EBP*) does penalise more the behavioral mismatches.

Altogether, these precision results show that the higher the k, the more the behavioral differences our measure can catch and penalise. Furthermore, in line with the time complexity analysis of MAP^k, we can see that when k increases, the execution time increases faster for those models allowing a huge amount of behavior, timing out for $k > 3$ and $k > 4$ on the *flower* and *all parallel* models (respectively).

In terms of ranking (see Table 34), our measure is the most consistent with the ranking of the models yielded by both *SD* (for acyclic models), *AA* (for all models), and EBP_p, than all other measures. As discussed above, the only differences are in the swapping of the order of the two models with self loops, and in the order of the *round robin* model. Note that given that both the round robin and the flower model have a value of zero in *AA* (and of 0.479 in EBP_p), the next model in the ranking (*all parallel*) is assigned a rank of 3 instead of 2 in MAP^k. This is just the way the ranking is computed and is not really indicative of a ranking inconsistency between the two measures. Another observation is that the ranking yielded by our precision measure remains the same for $k > 2$. This indicates that as we increase *k*, while the extent of behavioral differences we can identify and penalize increases, this is not achieved at the price of changing the ranking of the models.

6.4.4. Quantitative Evaluation Setup

In our second evaluation, we used the same public dataset of our benchmark (12 event logs) and eight proprietary logs sourced from several companies in the education, insurance, IT service management and IP management domains. For each of these 20 event logs, we automatically discovered three process models using SM, IM, and (SHM), totalling 60 log-model pairs that we used for our quantitative evaluation.

6.4.5. Quantitative Evaluation of MAF^k

We measured our MAF^k on each of the 60 log-model pairs, varying *k* in the range 2–5. For our comparison, we retained the *Alignment* fitness and PCC_2 fitness. For PCC we used a projections size of 2 as this was the highest value for which we were able to obtain more than 50% of the measurements. We held out *Token* fitness and *EBF* (both exact and partial matching variants), since they do not scale to real-life models, and we used *Alignment* as a baseline for our comparison, since it is to date the most-scalable and widely-accepted fitness measure for automated process discovery in real-life settings [13].

Table 35 and 36 show respectively the numerical values of each measurement and the fitness ranking of the models discovered by SM, IM and SHM for each log. [16] Also in this evaluation, we can observe that the values of MAF^k reduce as we increase the order *k*. This is by design, since the higher the *k* the more are the behavioral details captured by the measure, and the more are the mismatches identified by our measure. Furthermore, we recall that our MAF^k is very strict since it uses a boolean function to penalise behavioral mismatches.

---

[16] A "–" symbol is used to report a failed measurement due to either a time-out or an exception.

In Table 35, we can see that for the models discovered by SM from the logs PRT1 and PRT6, the values of MAF^k quickly drop when moving from $k = 3$ to $k = 4$. The results of *Alignment* and MAF^2 for the model of SM discovered from the log PRT9, are also interesting: *Alignment* returns a value of 0.915, suggesting an almost fully-fitting model, while our MAF^k returns a low value of fitness (0.197) already at $k = 2$. This can be linked to the fact that *Alignment* is sometimes too accommodative leading to counter-intuitive results, as shown in the qualitative evaluation.

If we look at the ranking yielded by the measures, MAF^k agrees with *Alignment* more than 30% of the times at $k = 3$ and $k = 5$, and more than 25% of the times at $k = 2$ and $k = 4$. Instead, PCC_2 agrees with *Alignment* only 10% of the times. Increasing k for our MAP^k does not alter frequently the ranking yielded: indeed 50% of the times the ranking was stable already at $k = 2$ and 75% at $k = 3$.

Finally, Tables 37 and 38 report the time performance of MAF^k, *Alignment* and PCC_2.[17] We separated the results by public and proprietary logs to allow the reproducibility of the experiments for the set of public logs. We note that PCC_2 underperforms *Alignment*, despite this measure was designed to be more efficient. Instead, MAF^k is significantly faster than *Alignment* in the majority of the log-model pairs, especially when the input model has a reasonable state space as that produced by SM and SHM (i.e. when flower-like constructs are absent). On the other hand, by increasing k the performance of MAF^k reduces sharply for flower-like models, as those produced by IM.

For the sake of measuring the time performance of *Alignment* we also considered the alternative implementation reported in [167]. However, this latter implementation consistently performed worse than the standard implementation for the majority of log-model pairs used in our evaluation.

6.4.6. Quantitative Evaluation of MAP^k

Similarly, we assessed our MAP^k against each real-life log-model pair while varying the order k in the range 2–5. Unfortunately, we were not able to use any of the reference measures used in the qualitative evaluation, because *SD* does not work for cyclic models (all models discovered by IM were cyclic) while *AA* and *EBP* (both exact and partial matching variants) do not scale to real-life models [2,166]. Furthermore, we excluded *NE* precision, since we were not able to perform the measurements for more than 50% of the log-model pairs within the two-hour timeout. Thus, we resorted to ETC_a and PCC_2 as two baselines.

Table 39 shows the results of the quantitative evaluation. In line with the former evaluation, the value of MAP^k decreases when k increases. However, being the behavior of the real-life models more complex than that of the artificial models, for some logs (e.g. the BPIC15 logs), it was not possible to compute MAP^4

---

[17] Highlighted in bold the best scores, underlined the second best scores.

Log	**BPIC12**			**BPIC13$_{cp}$**			**BPIC13$_{inc}$**			**BPIC14$_{f}$**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	0.970	0.990	-	0.989	0.820	0.940	0.977	0.920	0.910	0.767	0.890	-
PCC_2	0.043	0.043	0.687	0.480	0.308	0.696	0.000	0.035	0.000	0.407	0.106	0.261
MAF^2	0.400	0.840	0.896	0.636	0.364	0.636	0.667	0.667	0.583	0.441	0.971	1.000
MAF^3	0.308	0.834	0.715	0.565	0.217	0.478	0.548	0.613	0.419	0.338	0.796	0.972
MAF^4	0.267	0.834	0.416	0.568	0.091	0.364	0.429	0.586	0.271	0.212	0.626	0.903
MAF^5	0.257	0.370	0.197	0.537	0.061	0.256	0.356	0.557	0.188	0.154	0.465	0.788

Log	**BPIC15$_{1f}$**			**BPIC15$_{2f}$**			**BPIC15$_{3f}$**			**BPIC15$_{4f}$**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	0.900	0.970	-	0.773	0.948	0.981	0.780	0.950	0.950	0.731	0.955	0.991
PCC_2	0.598	0.266	0.000	0.015	0.087	0.013	0.523	0.024	0.016	0.000	0.000	0.007
MAF^2	0.758	0.960	0.992	0.538	0.853	0.919	0.413	0.894	0.894	0.543	0.921	0.974
MAF^3	0.614	0.949	0.990	0.335	0.793	0.885	0.224	0.804	0.811	0.329	0.842	0.959
MAF^4	0.510	0.934	0.983	0.226	0.688	0.855	0.147	0.643	0.734	0.230	0.558	0.938
MAF^5	0.434	0.535	0.901	0.168	0.307	0.825	0.105	0.290	0.678	0.169	0.372	0.916

Log	**BPIC15$_{5f}$**			**BPIC17$_{f}$**			**RTFMP**			**SEPSIS**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	0.791	0.937	1.000	0.962	0.979	0.954	1.000	0.980	0.980	0.763	0.991	0.920
PCC_2	0.005	0.301	0.016	0.244	0.466	0.013	0.000	0.062	0.757	0.000	0.129	0.296
MAF^2	0.591	0.922	0.987	0.935	0.935	0.903	0.257	0.843	0.957	0.226	0.930	0.757
MAF^3	0.371	0.874	0.982	0.923	0.885	0.769	0.187	0.743	0.930	0.155	0.953	0.678
MAF^4	0.263	0.347	0.981	0.904	0.843	0.675	0.151	0.655	0.889	0.164	0.962	0.654
MAF^5	0.191	0.135	0.982	0.909	0.818	0.606	0.125	0.576	0.882	0.187	0.647	0.631

Log	**PRT1**			**PRT2**			**PRT3**			**PRT4**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	0.977	0.902	0.883	0.811	-	-	0.824	0.975	1.000	0.833	0.927	1.000
PCC_2	0.276	0.225	0.883	0.000	0.000	0.802	0.264	0.420	0.056	0.082	0.309	0.318
MAF^2	0.459	0.730	0.811	0.291	0.987	0.747	0.581	0.977	0.977	0.541	0.892	1.000
MAF^3	0.318	0.418	0.691	0.111	0.880	0.637	0.291	0.882	0.976	0.284	0.622	1.000
MAF^4	0.258	0.154	0.585	0.071	0.803	0.640	0.185	0.807	0.982	0.154	0.318	1.000
MAF^5	0.223	0.035	0.502	0.077	0.354	0.684	0.131	0.778	0.985	0.084	0.118	1.000

Log	**PRT6**			**PRT7**			**PRT9**			**PRT10**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	0.943	0.989	1.000	0.910	1.000	1.000	0.915	0.900	0.964	0.969	0.964	-
PCC_2	0.632	0.320	0.503	0.426	0.385	0.533	0.780	0.011	0.990	0.327	0.623	-
MAF^2	0.545	1.000	1.000	0.605	1.000	1.000	0.197	0.818	0.833	0.471	0.966	-
MAF^3	0.220	0.898	1.000	0.243	1.000	1.000	0.106	0.596	0.652	0.374	0.925	-
MAF^4	0.082	0.776	1.000	0.161	1.000	1.000	0.074	0.312	0.476	0.343	0.934	-
MAF^5	0.044	0.716	1.000	0.141	1.000	1.000	0.055	0.127	0.349	0.343	0.852	-

Table 35: Comparison of fitness measures over the 20 real-life logs.

Log	**BPIC12**			**BPIC13$_{\text{cp}}$**			**BPIC13$_{\text{inc}}$**			**BPIC14$_{\text{f}}$**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	2	3	-	3	1	2	3	2	1	2	3	-
PCC_2	1	1	2	2	1	3	1	2	1	3	1	2
MAF^2	1	2	3	3	1	3	3	3	2	1	2	3
MAF^3	1	3	2	3	1	2	2	3	1	1	2	3
MAF^4	1	3	2	2	1	2	2	3	1	1	2	3
MAF^5	2	3	1	3	1	2	2	3	1	1	2	3

Log	**BPIC15$_{\text{1f}}$**			**BPIC15$_{\text{2f}}$**			**BPIC15$_{\text{3f}}$**			**BPIC15$_{\text{4f}}$**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	2	3	-	1	2	3	2	3	3	1	2	3
PCC_2	3	2	1	2	3	1	3	2	1	1	1	2
MAF^2	1	2	3	1	2	3	2	3	3	1	2	3
MAF^3	1	2	3	1	2	3	1	2	3	1	2	3
MAF^4	1	2	3	1	2	3	1	2	3	1	2	3
MAF^5	1	2	3	1	2	3	1	2	3	1	2	3

Log	**BPIC15$_{\text{5f}}$**			**BPIC17$_{\text{f}}$**			**RTFMP**			**SEPSIS**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	1	2	3	2	3	1	3	2	2	1	2	3
PCC_2	1	3	2	2	3	1	1	2	3	1	2	3
MAF^2	1	2	3	3	3	1	1	2	3	1	3	2
MAF^3	1	2	3	3	2	1	1	2	3	1	3	2
MAF^4	1	2	3	3	2	1	1	2	3	1	3	2
MAF^5	2	1	3	3	2	1	1	2	3	1	3	2

Log	**PRT1**			**PRT2**			**PRT3**			**PRT4**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	3	2	1	-	-	-	1	2	3	1	2	3
PCC_2	2	1	3	1	1	2	2	3	1	1	2	3
MAF^2	1	2	3	1	3	2	1	3	3	1	2	3
MAF^3	1	2	3	1	3	2	1	2	3	1	2	3
MAF^4	2	1	3	1	3	2	1	2	3	1	2	3
MAF^5	2	1	3	1	2	3	1	2	3	1	2	3

Log	**PRT6**			**PRT7**			**PRT9**			**PRT10**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
Alignment	1	2	3	2	3	3	2	1	3	3	2	-
PCC_2	3	1	2	2	1	3	2	1	3	1	2	-
MAF^2	2	3	3	1	3	3	1	2	3	2	3	-
MAF^3	1	2	3	1	3	3	1	2	3	2	3	-
MAF^4	1	2	3	1	3	3	1	2	3	2	3	-
MAF^5	1	2	3	1	3	3	1	2	3	2	3	-

Table 36: Models ranking yielded by fitness measures over the 20 real-life logs.

	Split Miner				**Inductive Miner**				**Struct. Heuristics Miner**			
Precision	**avg**	**max**	**min**	**total**	**avg**	**max**	**min**	**total**	**avg**	**max**	**min**	**total**
Alignment	30.1	217.8	0.5	361.0	33.7	155.2	0.5	404.8	37.2	174.4	0.7	335.0
PCC_2	77.1	685.1	0.1	925.4	138.9	1344.9	0.1	1667.1	152.1	1378.8	0.1	1825.1
MAF^2	0.2	2.0	>**0.1**	2.7	**0.3**	**1.1**	>**0.1**	**4.2**	**0.9**	**4.6**	>**0.1**	**10.7**
MAF^3	**0.1**	**0.3**	>**0.1**	**0.7**	108.2	890.9	>**0.1**	1298.1	4.2	27.9	>**0.1**	50.4
MAF^4	**0.1**	**0.3**	>**0.1**	0.9	1386.2	4383.9	>**0.1**	16634.9	14.1	84.7	>**0.1**	169.4
MAF^5	**0.1**	0.6	>**0.1**	1.3	1612.4	5365.6	>**0.1**	19348.6	39.1	214.4	>**0.1**	469.2

Table 37: Time performance (in seconds) of fitness measures using the twelve public logs.

	Split Miner				**Inductive Miner**				**Struct. Heuristics Miner**			
Precision	**avg**	**max**	**min**	**total**	**avg**	**max**	**min**	**total**	**avg**	**max**	**min**	**total**
Alignment	12.4	40.6	0.4	98.9	5.9	29.9	0.4	41.6	144.2	813.2	0.5	865.1
PCC_2	154.2	1226.2	>**0.1**	1233.3	257.0	2048.9	>**0.1**	2056.3	287.3	1960.2	>**0.1**	2010.8
MAF^2	0.2	1.0	>**0.1**	1.3	**2.1**	**15.8**	>**0.1**	**17.2**	**0.7**	**2.1**	>**0.1**	**4.8**
MAF^3	**0.1**	**0.3**	>**0.1**	**0.6**	8.7	66.2	>**0.1**	69.4	1.5	6.0	>**0.1**	10.4
MAF^4	**0.1**	0.4	>**0.1**	**0.6**	80.7	597.0	>**0.1**	645.2	12.5	52.6	>**0.1**	87.8
MAF^5	**0.1**	0.5	>**0.1**	0.7	352.8	2792.9	>**0.1**	2822.6	21.8	90.5	>**0.1**	152.7

Table 38: Time performance (in seconds) of fitness measures using the eight proprietary logs.

and MAP^5 for the models discovered by IM. This was due to scalability issues,[18] as the models discovered by IM exhibit flower-like behavior (with more than 50 distinct activities per flower construct), which is already identified by MAP^2 and MAP^3, whose values are very low for these models. We recall that by design, for small values of k, MAP^k compares small chunks of the model behavior to small chunks of the log behavior. Thus, low values of MAP^2 and MAP^3 already indicate poorly-precise models.

ETC_a and MAP^5 agree on the precision ranking 50% of the times (cf. Table 40), whilst ETC_a and PCC_2 agree on the 30% of the rankings. Also, in-line with the former evaluation, ETC_a is tolerant to infinite model behavior, regardless of the type of such behavior. An example that illustrates this flaw is the SEPSIS log. The models discovered by IM and SM are shown in Fig. 26 and 27. We observe that more than the 80% of the activities in the IM model are skippable and over 60% of them are inside a long loop, resembling a flower construct with some constraints, e.g. the first activity is always the same. Instead, the model discovered by SM, even if cyclic, does not allow many variants of behavior. Consequently, for the IM model, the value of MAP^k drastically drops when increasing k from 2 to 3, whilst it remains 1 for the SM model. In contrast, ETC_a gives a precision of 0.445 for IM, which is counter-intuitive considering its flower-like structure.

Finally, Tables 41 and 42 report the time performance of MAP^k, PCC_2 and ETC_a.[15] Similarly to the quantitative evaluation of fitness, we separated the results by public and proprietary logs to allow the reproducibility of the experiments for the public logs. The results are consistent with those obtained for fitness:

---

[18] The allocated RAM was not enough to complete the measurements.

Log	**BPIC12**			**BPIC13$_{\text{cp}}$**			**BPIC13$_{\text{inc}}$**			**BPIC14$_{\text{f}}$**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	0.762	0.502	-	0.974	1.000	0.992	0.979	0.558	0.978	0.673	0.646	-
PCC_2	1.000	0.902	0.919	0.935	0.347	0.703	1.000	0.588	0.938	0.990	0.744	0.991
MAP^2	1.000	0.399	0.316	1.000	1.000	0.708	1.000	1.000	1.000	1.000	0.786	0.500
MAP^3	0.826	0.083	0.073	0.952	0.944	0.682	0.965	0.955	0.976	1.000	0.701	0.274
MAP^4	0.640	0.013	0.027	0.931	0.917	0.638	0.919	0.898	0.911	1.000	0.656	0.164
MAP^5	0.362	-	0.015	0.907	0.925	0.626	0.893	0.828	0.883	1.000	0.631	0.162

Log	**BPIC15$_{\text{1f}}$**			**BPIC15$_{\text{2f}}$**			**BPIC15$_{\text{3f}}$**			**BPIC15$_{\text{4f}}$**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	0.880	0.566	-	0.901	0.556	0.594	0.939	0.554	0.671	0.910	0.585	0.642
PCC_2	0.995	0.885	1.000	1.000	0.912	1.000	0.979	0.974	1.000	1.000	1.000	1.000
MAP^2	1.000	0.144	0.054	1.000	0.160	0.995	1.000	0.196	1.000	1.000	0.121	1.000
MAP^3	0.969	0.019	0.016	0.972	0.022	0.835	0.986	0.032	0.787	0.963	0.015	0.789
MAP^4	0.932	-	-	0.930	-	0.588	0.967	-	0.502	0.920	-	0.531
MAP^5	0.896	-	-	0.878	-	0.356	0.939	-	0.275	0.877	-	0.321

Log	**BPIC15$_{\text{5f}}$**			**BPIC17$_{\text{f}}$**			**RTFMP**			**SEPSIS**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	0.943	0.179	0.687	0.846	0.699	0.620	1.000	0.700	0.952	0.859	0.445	0.419
PCC_2	0.998	0.913	1.000	0.939	0.600	0.911	1.000	0.831	0.793	1.000	0.745	0.689
MAP^2	1.000	0.092	1.000	1.000	0.843	0.347	1.000	0.939	0.745	1.000	0.603	0.521
MAP^3	0.964	0.006	0.817	0.705	0.566	0.151	0.955	0.492	0.309	0.954	0.210	0.193
MAP^4	0.917	-	0.577	0.482	0.370	0.069	0.862	0.182	0.087	0.895	0.048	0.062
MAP^5	0.858	-	0.366	0.340	0.245	0.034	0.756	0.070	0.025	0.831	-	-

Log	**PRT1**			**PRT2**			**PRT3**			**PRT4**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	0.985	0.673	0.768	0.737	-	-	0.914	0.680	0.828	0.995	0.753	0.865
PCC_2	0.970	0.607	0.793	1.000	0.788	0.617	0.945	0.716	0.996	0.955	0.705	0.822
MAP^2	1.000	0.868	0.842	1.000	0.975	1.000	1.000	0.913	1.000	1.000	0.917	1.000
MAP^3	0.966	0.729	0.655	0.995	0.960	0.992	0.907	0.878	0.954	1.000	0.979	1.000
MAP^4	0.939	0.677	0.467	0.995	0.631	0.895	0.839	0.763	0.631	1.000	0.977	1.000
MAP^5	0.917	0.658	0.309	0.979	0.442	0.410	0.775	0.560	0.293	0.998	0.982	0.968

Log	**PRT6**			**PRT7**			**PRT9**			**PRT10**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	1.000	0.822	0.908	0.999	0.726	0.998	0.999	0.611	0.982	0.972	0.790	-
PCC_2	0.800	0.694	0.832	0.857	0.743	0.857	0.927	0.658	0.738	1.000	0.804	-
MAP^2	1.000	0.957	1.000	1.000	0.927	1.000	1.000	0.991	0.958	1.000	0.387	-
MAP^3	1.000	0.873	0.983	1.000	0.920	0.972	0.971	0.526	0.542	0.814	0.061	-
MAP^4	1.000	0.830	0.950	1.000	0.699	0.725	0.913	0.207	0.208	0.639	0.007	-
MAP^5	1.000	0.574	0.632	1.000	0.543	0.645	0.853	0.076	0.064	0.435	-	-

Table 39: Comparison of precision measures over 20 real-life logs.

Log	**BPIC12**			**BPIC13$_{\text{cp}}$**			**BPIC13$_{\text{inc}}$**			**BPIC14$_{\text{f}}$**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	3	2	-	1	3	2	3	1	2	3	2	-
PCC_2	3	1	2	3	1	2	3	1	2	2	1	3
MAP^2	3	2	1	3	3	2	3	3	3	3	2	1
MAP^3	3	2	1	3	2	1	2	1	3	3	2	1
MAP^4	3	1	2	3	2	1	3	1	2	3	2	1
MAP^5	3	-	2	2	3	1	3	1	2	3	2	1

Log	**BPIC15$_{\text{1f}}$**			**BPIC15$_{\text{2f}}$**			**BPIC15$_{\text{3f}}$**			**BPIC15$_{\text{4f}}$**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	3	2	-	3	1	2	3	1	2	3	1	2
PCC_2	2	1	3	3	2	3	2	1	3	3	3	3
MAP^2	3	2	1	3	1	2	3	2	3	3	2	3
MAP^3	3	2	1	3	1	2	3	1	2	3	1	2
MAP^4	-	-	-	3	-	2	3	-	2	3	-	2
MAP^5	-	-	-	3	-	2	3	-	2	3	-	2

Log	**BPIC15$_{\text{5f}}$**			**BPIC17$_{\text{f}}$**			**RTFMP**			**SEPSIS**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	3	1	2	3	2	1	3	1	2	3	2	1
PCC_2	2	1	3	3	1	2	3	2	1	3	2	1
MAP^2	3	2	3	3	2	1	3	2	1	3	2	1
MAP^3	3	1	2	3	2	1	3	2	1	3	2	1
MAP^4	3	-	2	3	2	1	3	2	1	3	1	2
MAP^5	3	-	2	3	2	1	3	2	1	-	-	-

Log	**PRT1**			**PRT2**			**PRT3**			**PRT4**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	3	1	2	-	-	-	3	1	2	3	1	2
PCC_2	3	1	2	3	2	1	2	1	3	3	1	2
MAP^2	3	2	1	3	2	3	3	2	3	3	2	3
MAP^3	3	2	1	3	1	2	2	1	3	3	2	3
MAP^4	3	2	1	3	1	2	3	2	1	3	2	3
MAP^5	3	2	1	3	2	1	3	2	1	3	2	1

Log	**PRT6**			**PRT7**			**PRT9**			**PRT10**		
Miner	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**	**SM**	**IM**	**SHM**
ETC_a	3	1	2	3	1	2	3	1	2	3	2	-
PCC_2	2	1	3	3	2	3	3	1	2	3	2	-
MAP^2	3	2	3	3	2	3	3	2	1	3	2	-
MAP^3	3	1	2	3	1	2	3	1	2	3	2	-
MAP^4	3	1	2	3	1	2	3	1	2	3	2	-
MAP^5	3	1	2	3	1	2	3	2	1	-	-	-

Table 40: Models ranking yielded by precision measures over 20 real-life logs.

	Split Miner				Inductive Miner				Struct. Heuristics Miner			
Precision	**avg**	**max**	**min**	**total**	**avg**	**max**	**min**	**total**	**avg**	**max**	**min**	**total**
ETC_a	60.0	351.9	0.3	720.3	84.2	642.7	0.1	1009.8	34.0^{+}	101.4^{+}	0.2^{+}	305.9^{+}
PCC_2	145.8	1482.8	0.1	1749.6	157.3	1598.3	0.1	1887.3	164.8	1530.4	0.1	1977.8
MAP^2	**0.1**	0.8	>**0.1**	1.6	**0.9**	**3.3**	>**0.1**	**10.9**	**1.1**	**5.7**	>**0.1**	**12.9**
MAP^3	**0.1**	**0.4**	>**0.1**	**1.5**	323.2	2844.0	>**0.1**	3878.9	8.4	43.2	>**0.1**	100.4
MAP^4	0.2	0.6	>**0.1**	1.9	818.5^{+}	4377.6^{+}	>**0.1**$^{+}$	5729.3^{+}	49.8	340.2	>**0.1**	547.6
MAP^5	0.7	3.2	>**0.1**	8.7	-	-	-	-	97.2^{+}	357.4^{+}	>**0.1**$^{+}$	972.4^{+}

Table 41: Time performance (in seconds) of precision measures using the twelve public logs ('+' indicates a result obtained on a subset of the twelve logs, due to some of the measurements not being available).

	Split Miner				Inductive Miner				Struct. Heuristics Miner			
Precision	**avg**	**max**	**min**	**total**	**avg**	**max**	**min**	**total**	**avg**	**max**	**min**	**total**
ETC_a	16.1	106.5	0.2	129.1	16.4	99.2	0.2	114.9	74.3	350.2	0.7	520.1
PCC_2	184.8	1468.9	>**0.1**	1478.8	152.7	1213.9	>**0.1**	1221.3	242.6	1649.4	>**0.1**	1698.5
MAP^2	**0.1**	0.4	>**0.1**	0.6	**2.1**	**15.9**	>**0.1**	**16.7**	**0.7**	**2.3**	>**0.1**	**4.7**
MAP^3	**0.1**	**0.2**	>**0.1**	**0.5**	7.0	47.0	>**0.1**	55.9	2.7	12.6	>**0.1**	19.1
MAP^4	**0.1**	0.4	>**0.1**	0.7	146.2	1131.2	>**0.1**	1169.9	29.4	120.0	>**0.1**	205.9
MAP^5	0.8	5.8	>**0.1**	6.8	55.5	332.6	0.1	388.2	77.8	329.4	0.1	544.4

Table 42: Time performance (in seconds) of precision measures using the eight proprietary logs.

PCC_2 is the slowest measure while MAP^k is overall the fastest measure, especially for those models that do not exhibit flower-like structures (SM and SHM), while ETC_a outperforms our measure when the state space of the model is very large (in the case of models discovered by IM, for high values of k).

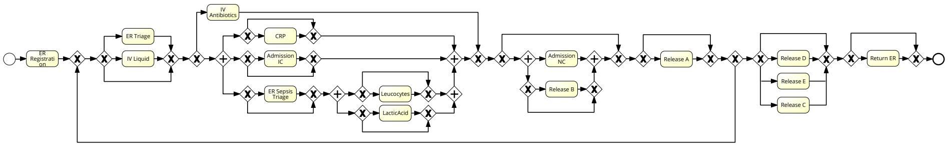


Figure 26: Model discovered by IM from the SEPSIS log.

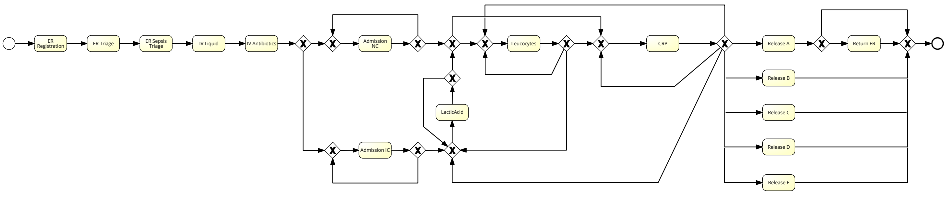


Figure 27: Model discovered by SM from the SEPSIS log.

6.4.7. The Role of k

As expected, the results of both the qualitative and the quantitative evaluation show that the order k directly influences how our Markovian fitness and precision measures penalise behavioral mismatches between log and model, and ultimately how they rank process models by accuracy. Furthermore, k plays an important role on the time performance of our proposed measures. Indeed, while for low k's MAF^k and MAP^k scale well to large real-life logs, outperforming the state of the art, for high k's, the time performance in some cases deteriorates even dramatically. However, we showed that in practice a low value of k does approximate well the fitness and precision ranking that would be obtained with hight k values, over a set of automatically discovered process models.

We remind that $k+1$ defines the size of the subtraces in the log and in the process model that we compare with each other to detect mismatches. Bearing this in mind, we advise that whenever possible k should be set to the length of the longest trace recorded in the log, in order to obtain the most accurate results.

6.5. Summary

In this chapter, we presented a family of fitness and precision measures to assess the accuracy of process models automatically discovered from event logs. The Markovian fitness and precision measures compare the k^{th}-order Markovian abstraction of a process model and that of an event log using a graph matching algorithm. We showed that the fitness measures fulfill six out of the seven fitness properties proposed by Syring et al. [4] for any value of k and all seven properties for a suitably chosen value of k dependent on the log. Similarly, the precision measures fulfil seven of the ten precision properties proposed by Tax et al. [5] and Syring et al. [4] for any value of k and all ten properties for a value of k dependent on the log.

While fulfilling the proposed properties is a desirable quality, it does not guarantee that the proposed measures provide intuitive results in practice. To validate the intuitiveness of the proposed measures, we compared the ranking they induce against those induced by existing fitness and precision measures using a collection of model-log pairs proposed by van Dongen et al. [2] (extended to cover fitness in addition to precision). For $k \geq 4$, the proposed fitness measures induce rankings that coincide with alignment-based fitness – a commonly used fitness measure. Meanwhile, for $k \geq 3$, the proposed precision measures induce rankings consistent with those of the anti-alignment precision measure, which has been previously posited as a ground-truth for precision measures.

A second evaluation using real-life event logs showed that the execution times of the proposed fitness measures (for $k \leq 5$) are considerably lower than existing fitness measures, except when applied to process models that contain flower structures, in which case alignment-based fitness offers the best performance. Similarly, the execution times of the proposed precision measures (for $k \leq 5$) are considerably lower than existing precision measures, except for models that contain flower structures in which case ETC_a precision provides the best performance.

Despite such a result may limit the Markovian fitness and precision and their applicability in a conformance checking scenario, we recall that our goal was to design scalable accuracy measures for improving the exploration of the solution space of an APDA when we aim at optimizing the accuracy of its discovered process model. In the latter setting, the Markovian fitness and precision perform very well (as we show in the next chapter), because regardless of the value of *k*, the comparison between the process model and the event log Markovian abstractions allows us to identify the behaviour to add to (remove from) the process model in order to improve its fitness (precision). Indeed, such information is encoded in the edges of the event log Markovian abstraction that cannot be found in the process model Markovian abstraction (and viceversa), and it is independent from the value of *k* and not prone to approximation by design.

7. OPTIMIZED AUTOMATED PROCESS DISCOVERY

The results of our benchmark (see Chapter 4) suggested that the accuracy of APDAs can be enhanced via hyper-parameter optimization. On the other hand, such an exercise is computationally heavy, requiring execution times in the order of hours when applied to real-life logs. As we mentioned in the previous chapter, this inefficiency relates to two different problems: (i) low scalability of the accuracy measures, especially precision; and (ii) a blind exploration of the solution space. While in the previous chapter we designed fast accuracy measures to tackle the former problem, in this chapter we present a general framework that allows an APDA to efficiently explore its solution space, with the precise goal of answering our **RQ3**.[1]

In order to discover optimal[2] process models, early research studies proposed APDAs based on population-based metaheuristics, chiefly genetic algorithms [149, 168], and single-solution-based metaheuristics such as simulated annealing [169, 170], which are less computationally demanding. However, these studies did not address the problem of optimizing the accuracy of an APDA, but rather they proposed a new APDA based on optimization metaheuristics.

In this chapter, [3] we introduce a flexible framework to enhance APDAs by applying different optimization single-solution-based metaheuristics. The core idea is to perturb the intermediate representation of an event log used by the majority of the available APDAs, namely the Directly-follows Graph (DFG). Specifically, we consider perturbations that add or remove edges with the aim of improving fitness or precision, and allowing the underlying APDA to discover a process model from the perturbed DFG.

We instantiated our framework for three state-of-the-art APDAs: Split Miner, Fodina, and Inductive Miner, and using a benchmark of 20 real-life logs, we compared the accuracy gains yielded by four optimization metaheuristics relative to each other and relative to the hyper-parameter optimized APDA. The experimental evaluation also considers the impact of metaheuristic optimization on model complexity measures as well as on execution times.

The remaining of this chapter is structured as follows. Section 7.1 gives an overview of the state-of-the-art optimization metaheuristics and their application to the problem of automated process discovery. Section 7.2 presents the proposed metaheuristic optimization framework and its instantiations. Section 7.3 reports on the empirical evaluation of the three instantiation of our framework. Finally, Section 7.4 summarises the chapter.

---

[1] *How can the accuracy of an automated process discovery approach be efficiently optimized?*

[2] In terms of maximizing one or more quality dimensions.

[3] Corresponding to [21].

7.1. Optimization Metaheuristics

The term *optimization metaheuristics* refers to a parameterized algorithm, which can be instantiated to address a wide range of optimization problems. Metaheuristics are usually classified into two broad categories [171]: i) *single-solution-based metaheuristics*, or S-metaheuristics, which explore the solution space one solution at a time starting from a single initial solution of the problem; and ii) *population-based metaheuristics*, or P-metaheuristics, which explore a population of solutions generated by mutating, combining, and/or improving previously identified solutions. S-metaheuristics tend to converge faster towards an optimal solution (either local or global) than P-metaheuristics, since the latter by dealing with a set of solutions require more time to assess and improve the quality of each single solution. P-metaheuristics are computationally heavier than S-metaheuristics, but they are more likely to escape local optima. Since an exhaustive discussion on all the available metaheuristics is beyond the scope of this thesis, in the following, we focus only on the S-metaheuristics that are relevant to our context.

Iterated Local Search [172] starts from a (random) solution and explores the neighbouring solutions (i.e. solutions obtained by applying a change to the current solution) in search of a better one. When a better solution cannot be found, it perturbs the current solution and starts again. The perturbation is meant to avoid local optimal solutions. The exploration of the solution-space ends when a given termination criterion is met (e.g. maximum iterations, timeout).

Tabu Search [173] is a memory-driven local search. Its initialization includes a (random) solution and three memories: short, intermediate, and long term. The short-term memory keeps track of recent solutions and prohibits to revisit them. The intermediate-term memory contains criteria driving the search towards the best solutions. The long-term memory contains characteristics that have often been found in many visited solutions, to avoid revisiting similar solutions. Using these memories, the neighbourhood of the initial solution is explored and a new solution is selected accordingly. The solution-space exploration is repeated until a termination criterion is met.

Simulated Annealing [174] is based on the concepts of Temperature (T, a parameter chosen arbitrarily) and Energy (E, the objective function to minimize). At each iteration the algorithm explores (some of) the neighbouring solutions and compares their energies with the one of the current solution. This latter is updated if the energy of a neighbour is lower, or with a probability that is function of T and the energies of the current and candidate solutions, usually $e^{-\frac{|E_1-E_2|}{T}}$. The temperature drops over time, thus reducing the chance of updating the current solution with a higher-energy one. The algorithm ends when a termination criterion is met, which often relates to the energy or the temperature (e.g. energy below a threshold or $T = 0$).

Evolutionary (Genetic) Algorithms [175, 176] are inspired by Darwin's theory of evolution. Starting from a set of (random) solutions, a new solution is generated by mixing characteristics of two parents selected from the set of the current solutions, such an operation is known as *crossover*. Subsequently, *mutations* are applied to the new solutions to introduce randomness and avoid local optimal solution. Finally, the solutions obtained are assessed and a subset is retained for the next iteration. The algorithm continues until a stop criterion is met.

Swarm Particle Optimization [177] starts from a set of (random) solutions, referred to as particles. Each particle is identified using the concepts of *position* and *velocity*. The position is a proxy for the particle qualities and it embeds the characteristics of the solution, while the velocity is used to alter the position of the particles at each iteration. Furthermore, each particle has memory of its best position encountered during the roaming of the search space, as well as the best position encountered by any other particle. At each iteration, the algorithm updates the particles positions according to their velocities and updates the best positions found. When a termination condition is met, the algorithm returns the particle having the absolute best position among the whole swarm.

Imperialist Competitive Algorithm [178] is inspired by the historical colonial period. It starts from a (random) set of solutions, called countries. Each country is assessed via an objective function, and a subset is selected as imperialistic countries (the selection is based on their objective function scores). All the countries left (i.e. those having low objective function scores) are considered colonies of the closest (by characteristics) imperialistic country. Then, each colony is altered to resemble its imperialistic country, the objective function scores are re-computed, and the colonies that became better than their imperialistic country are promoted to imperialistic countries and vice-versa. When a termination condition is met, the country with the highest objective function score is selected as the best solution.

Optimization metaheuristics have been considered in a few previous studies on automated process discovery. An early attempt to apply P-metaheuristics to automated process discovery was the Genetic Miner proposed by De Medeiros [149], subsequently overtaken by the Evolutionary Tree Miner [168]. Other applications of P-metaheuristics to automated process discovery are based on the imperialist competitive algorithm [179] and the particle swam optimization [180]. In our context, the main limitation of P-metaheuristics is that they are computationally heavy due to the cost of constructing a solution (i.e. a process model) and evaluating its accuracy. This leads to execution times in the order of hours, to converge to a solution that in the end is comparable to those obtained by state-of-the-art APDAs that do not rely on optimization metaheuristics (see Chapter 4). Finally, a handful of studies have considered the use of S-metaheuristics in our context, specifically simulated annealing [169, 170], but these proposals are preliminary and have not been compared against state-of-the-art approaches on real-life logs.

7.2. Metaheuristic Optimization Framework

This section outlines our framework for optimizing APDAs by means of S-metaheuristics (cf. Section 7.1). First, we give an overview of the framework and its core components. Next, we discuss the adaptation of the S-metaheuristics to the problem of process discovery. Finally, we describe the instantiations of our framework for Split Miner, Fodina, and Inductive Miner.

7.2.1. Preliminaries

In order to discover a process model, an APDA takes as input an event log and transforms it into an intermediate representation from which a process model is derived. In many APDAs, such an intermediate representation is the DFG, which can be represented as a numerical matrix as formalized below.

Definition 7.2.1. ***[Event Log]*** *Given a set of activities* $\mathscr{A}$, *an* event log $\mathscr{L}$ *is a multiset of traces where a trace* $t \in \mathscr{L}$ *is a sequence of activities* $t = \langle a_1, a_2, \ldots, a_n \rangle$, *with* $a_i \in \mathscr{A}, 1 \leq i \leq n$.

Definition 7.2.2. ***[Directly-Follows Graph (DFG) of an Event Log]*** *Given an event log* $\mathscr{L}$, *its* Directly-Follows Graph (DFG) *is a directed graph* $\mathscr{G}_{\mathscr{L}} = (N, E)$, *where:* N *is the set of nodes,* $N = \{a \in \mathscr{A} \mid \exists t \in \mathscr{L} \land a \in t\}$; *and* E *is the set of edges* $E = \{(x,y) \in N \times N \mid \exists t = \langle a_1, a_2, \ldots, a_n \rangle, t \in \mathscr{L} \land a_i = x \land a_{i+1} = y [1 \leq i \leq n-1]\}$.

Definition 7.2.3. ***[Refined DFG]*** *Given an event log* $\mathscr{L}$ *and its DFG* $\mathscr{G}_{\mathscr{L}} = (N, E)$, *a* Refined (DFG) *is a directed graph* $\mathscr{G} = (N', E')$, *where:* $N' \subseteq N$ *and* $E' \subseteq E$. *If* $N' = N$ *and* $E' = E$, *the refined DFG is equivalent to the event log DFG.*

Definition 7.2.4. ***[DFG-Matrix]*** *Given a refined DFG* $\mathscr{G} = (N, E)$ *and a function* $\theta : N \rightarrow [1, |N|]$,[4] *the DFG-Matrix is a squared matrix* $X_{\mathscr{G}} \in [0,1] \cap \mathbb{N}^{|N| \times |N|}$, *where each cell* $x_{i,j} = 1 \iff \exists (a_1, a_2) \in E \mid \theta(a_1) = i \land \theta(a_2) = j$, *otherwise* $x_{i,j} = 0$.

We say that an APDA is *DFG-based* if it first generates the DFG of an event log, then applies one or more algorithms to analyse and/or manipulate the DFG (e.g. a filtering algorithm), and finally exploits the refined DFG to produce the final process model. Examples of DFG-based APDAs are Inductive Miner [152], Heuristics Miner [8, 42], Fodina [12], and Split Miner (see Chapter 5).

Different DFG-based APDAs can extract different refined DFGs from the same log. Also, a DFG-based APDA may discover different refined DFGs from the same log depending on its hyperparameter settings (e.g. a filtering threshold). The algorithm(s) used by a DFG-based APDA to discover the refined DFG from the event log and convert it into a process model may greatly affect the accuracy of an APDA. Accordingly, our framework focuses on optimizing the discovery of the refined DFG rather than its conversion into a process model.

---

[4] θ maps each node of the DFG to a natural number.

For simplicity, in the remainder of this chapter, when using the term DFG we refer to the refined DFG.

7.2.2. Approach Overview

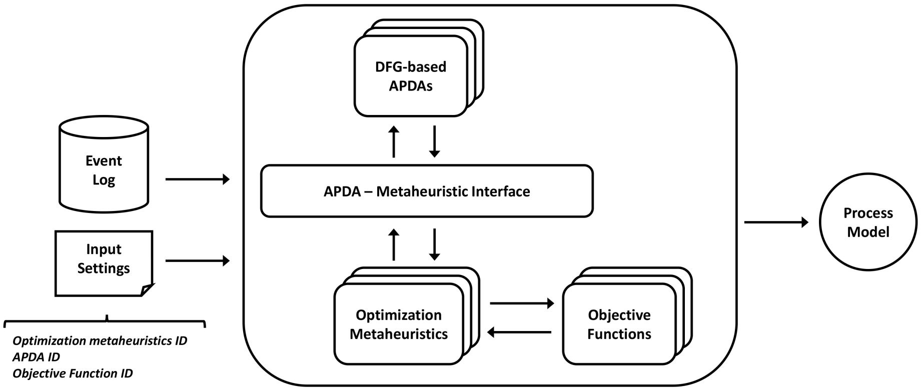


Figure 28: Overview of our approach.

As shown in Figure 28, our approach takes three inputs (in addition to the log): i) the optimization metaheuristics; ii) the objective function to be optimized (e.g. F-score); iii) and the DFG-based APDA to be used for discovering a process model.

Algorithm 12 describes how our approach operates. First, the input event log is given to the APDA, which returns the discovered DFG and its corresponding process model (lines 1 and 2). This DFG becomes the *current DFG*, whilst the process model becomes the *best process model* (so far). This process model's objective function score (e.g. the F-score) is stored as the current score and the best score (lines 3 and 4). The current DFG is then given as input to the function *GenerateNeighbours*, which applies changes to the current DFG to generate a set of neighbouring DFGs (line 6). The latter ones are given as input to the APDA, which returns the corresponding process models. The process models are assessed by the objective function evaluators (line 9 to 13). When the metaheuristic receives the results from the evaluators (along with the current DFG and its score), it chooses the new current DFG and updates the current score (lines 14 and 15). If the new current score is higher than the best score (line 16), it updates the best process model and the best score (lines 17 and 18). After the update, a new iteration starts, unless a termination criterion is met (e.g. a timeout, a maximum number of iterations, or a minimum threshold for the objective function). In the latter case, the framework outputs the best process model identified, i.e. the process model scoring the highest value for the objective function.

7.2.3. Adaptation of the Optimization Metaheuristics

To adapt Iterative Local Search (ILS), Tabu Search (TABU), and Simulated Annealing (SIMA) to the problem of automated process discovery, we need to define

Algorithm 12: Optimization Approach

```
    input : Event Log $\mathscr{L}$, Metaheuristic $\omega$, Objective Function $\mathscr{F}$ , DFG-based APDA $\alpha$
1   CurrentDFG $\mathscr{G}_c \leftarrow$ DiscoverDFG($\alpha$, $\mathscr{L}$);
2   BestModel $\hat{m} \leftarrow$ ConvertDFGtoProcessModel($\alpha$, $\mathscr{G}_c$);
3   CurrentScore $s_c \leftarrow$ AssessQuality($\mathscr{F}$, $\hat{m}$);
4   BestScore $\hat{s} \leftarrow s_c$;
5   while CheckTerminationCriteria() = FALSE do
6       Set $V \leftarrow$ GenerateNeighbours($\mathscr{G}_c$);
7       Map $S \leftarrow \varnothing$;
8       Map $M \leftarrow \varnothing$;
9       for $\mathscr{G} \in V$ do
10          ProcessModel $m \leftarrow$ ConvertDFGtoProcessModel($\alpha$, $\mathscr{G}$);
11          Score $s \leftarrow$ AssessQuality($\mathscr{F}$, $m$);
12          add ($\mathscr{G}$, $s$) to $S$;
13          add ($\mathscr{G}$, $m$) to $M$;
14      $\mathscr{G}_c \leftarrow$ UpdateDFG($\omega$, $S$, $\mathscr{G}_c$, $s_c$);
15      $s_c \leftarrow$ GetMapElement($S$, $\mathscr{G}_c$);
16      if $\hat{s} < s_c$ then
17          $\hat{s} \leftarrow s_c$;
18          $\hat{m} \leftarrow$ GetMapElement($M$, $\mathscr{G}_c$);
19  return $\hat{m}$;
```

the following three concepts: i) the problem solution space; ii) a solution neighbourhood; iii) the objective function. These design choices influence how each of the metaheuristics navigates the solution space and escapes local minima, i.e. how to design the Algorithm 12 functions: *GenerateNeighbours* and *UpdateDFG*, resp. lines 6 and 14.

Solution Space. Since our goal is the optimization of APDAs, we are forced to choose a solution space that fits well our context regardless the selected APDA. If we assume that the APDA is DFG-based (that is the case for the majority of the available APDAs), we can define the solution space as the set of all the DFG discoverable from the event log. Indeed, any DFG-based APDA can generate deterministically a process model from a DFG.

Solution Neighbourhood. Having defined the solution space as the set of all the DFG discoverable from the event log, we can refer to any element of this solution space as a DFG-Matrix. Given a DFG-Matrix, we define its neighbourhood as the set of all the matrices having one different cell value (i.e. DFGs having one more/less edge). In the following, every time we refer to a DFG we assume it is represented as a DFG-Matrix.

Objective Function. It is possible to define the objective function as any function assessing one of the four quality dimensions for discovered process models (introduced in Chapter 2). However, given that we are interested in optimizing the APDAs to discover the most accurate process model, in our optimization framework instantiations we refer to the objective function as the F-score of fitness and precision. Nonetheless, we remark that our approach can operate also with objective functions that take into account multiple quality dimensions striving for a trade-off, e.g. F-score and model complexity.

Having defined the solution space, a solution neighbourhood, and the objective function, we can turn our attention to how ILS, TABU, and SIMA navigate the solution space. ILS, TABU, and SIMA share similar traits in solving an optimization problem, especially when it comes to the navigation of the solution space. Given a problem and its solution space, any of these three S-metaheuristics starts from a (random) solution, discovers one or more neighbouring solutions, and assesses them with the objective function to find a solution that is better than the current one. If a better solution is found, it is chosen as the new current solution and the metaheuristic performs a new neighbourhood exploration. If a better solution is not found, e.g. the current solution is locally optimal, the three metaheuristics follow different approaches to escape the local optimum and continue the solution space exploration. Algorithm 12 orchestrates and facilitates the parts of this procedure shared by the three metaheuristics. However, we must define the functions *GenerateNeighbours* (GNF) and *UpdateDFG* (UDF).

The GNF receives as input a solution of the solution space, i.e. a DFG, and it generates a set of neighbouring DFGs. By definition, GNF is independent from the metaheuristic and it can be as simple or as elaborate as we demand. An example of a simple GNF is a function that randomly selects neighbouring DFGs turning one cell of the input DFG-Matrix to 0 or to 1. Whilst, an example of an elaborate GNF is a function that accurately selects neighbouring DFGs relying on the feedback received from the objective function assessing the input DFG, as we show in Section 7.2.4.

The UDF is at the core of our optimization, and it represents the metaheuristic itself. It receives in input the neighbouring DFGs, the current DFG, and the current score, and it selects among the neighbouring DFGs the one that should become the new current DFG. At this point, we can differentiate two cases: i) among the input neighbouring DFGs there is at least one having a higher objective function score than the current; ii) none of the input neighbouring DFGs has a higher objective function score than the current. In the first case, UDF always outputs the DFG having the highest score (regardless the selected metaheuristic). In the second case, the current DFG may be a local optimum, and each metaheuristic escapes it with a different strategy.

Iterative Local Search applies the simplest strategy, it perturbs the current DFG. The perturbation is meant to alter the DFG in such a way to escape the local optimum, e.g. randomly adding and removing multiple edges from the current DFG. The perturbed DFG is the output of the UDF.

Tabu Search relies on its three memories to escape a local optimum. The short-term memory (a.k.a. Tabu-list), containing DFG that must not be explored further. The intermediate-term memory, containing DFGs that should lead to better results and, therefore, should be explored in the near future. The long-term memory, containing DFGs (with characteristics) that have been seen multiple times and, therefore, not to explore in the near future. TABU updates the three memories each time the UDF is executed. Given the set of neighbouring DFGs and their respective objective function scores (see Algorithm 12, map S), TABU adds each DFG to a different memory. DFGs worsening the objective function score are added to the Tabu-list. DFGs improving the objective function score, yet less than another neighbouring DFG, are added to the intermediate-term memory. DFGs that do not improve the objective function score are added to the long-term memory. Also, the current DFG is added to the Tabu-list, since it is already explored. When TABU does not find a better DFG in the neighbourhood of the current DFG, it returns the latest DFG added to the intermediate-term memory. If the intermediate-term memory is empty, TABU returns the latest DFG added to the long-term memory. If both these memories are empty, TABU requires a new (random) DFG from the APDA, and outputs its DFG.

Simulated Annealing avoids getting stuck in a local optimum by allowing the selection of DFGs worsening the objective function score. In doing so, SIMA explores areas of the solution space that other S-metaheuristics do not. When a better DFG is not found in the neighbourhood of the current DFG, SIMA analyses one neighbouring DFG at a time. If this latter does not worsen the objective function score, SIMA outputs it. Instead, if the neighbouring DFG worsens the objective function score, SIMA outputs it with a probability of $e^{-\frac{|s_n - s_c|}{T}}$, where s_n and s_c are the objective function scores of (respectively) the neighbouring DFG and the current DFG, and the temperature T is an integer that converges to zero as a linear function of the maximum number of iterations. The temperature is fundamental to avoid updating the current DFG with a worse one if there would be no time to recover from the worsening (i.e. too few iterations left for continuing the exploration of the solution space from the worse DFG).

7.2.4. Framework Instantiation

To assess our framework, we instantiated it for three APDAs: Split Miner, Fodina, and Inductive Miner. These three APDAs are all DFG-based, and they are representatives of the state-of-the-art (see Chapter 4 and 5).

To complete the instantiation of our framework for any concrete DFG-based APDA, it is necessary to implement an interface that allows the metaheuristics to interact with the APDA (as discussed above). Such an interface should provide four functions: *DiscoverDFG* and *ConvertDFGtoProcessModel* (see Algorithm 12), the *Restart Function* (RF) for TABU, and the *Perturbation Function* (PF) for ILS.

The first two functions, *DiscoverDFG* and *ConvertDFGtoProcessModel*, are inherited from the DFG-based APDA, in our case Split Miner, Fodina, and Inductive Miner. We note that, Split Miner and Fodina receive as input hyperparameter settings that can vary the output of the *DiscoverDFG* function. Precisely, *Split Miner* has two hyperparameters: the noise filtering threshold, used to drop infrequent edges in the DFG, and the parallelism threshold, used to determine which potential parallel relations between activities are used when discovering the process model from the DFG. Whilst *Fodina* has three hyperparameters: the noise filtering threshold, similar to the one of Split Miner, and two threshold to detect respectively self-loops and short-loops in the DFG. Instead, the variant of Inductive Miner [152] we used in our optimization framework[5] does not have any input hyperparameters.

To discover the initial DFG (Algorithm 12, line 1) with Split Miner we use its default parameters. We removed the randomness for discovering the initial DFG because most of the times, the DFG discovered by Split Miner with default parameters is already a good solution (see Chapter 5), and starting the solution space exploration from this latter can reduce the total exploration time.

Similarly, if Fodina is the selected APDA, the initial DFG (Algorithm 12, line 1) is discovered using the default parameters of Fodina, even though there is no guarantee that the default parameters allow Fodina to discover a good starting solution (see 2). Yet, this design choice is less risky than randomly choose input hyperparameters to discover the initial DFG, because it is likely Fodina would discover unsound models when randomly tuned, given that it does not guarantee soundness.

On the other hand, Inductive Miner (in its simplest variant [152]) does not apply any manipulation to the discovered initial DFG, in this case, we (pseudo) randomly generate an initial DFG. Differently than the case of Fodina, this is a suitable design choice for Inductive Miner, because it always guarantees block-structured sound process models, regardless of the DFG.

Function RF is very similar to *DiscoverDFG*, since it requires the APDA to output a DFG, the only difference is that RF must output a different DFG every time it is executed. We adapted the *DiscoverDFG* function of Split Miner and Fodina to output the DFG discovered with default parameters the first time it is executed, and a DFG discovered with (pseudo)random parameters for the following executions. The case of Inductive Miner is simpler, because the *DiscoverDFG* function always returns a (pseudo)random DFG. Consequently, we mapped RF to the *DiscoverDFG* function.

Finally, function PF can be provided either by the APDA (through the interface) or by the metaheuristic. However, PF can be more effective when not generalised by the metaheuristic, allowing the APDA to apply different perturbations to the DFGs, taking into account how the APDA converts the DFG to a process model. We chose a different PF for each of the three APDAs.

Split Miner PF. We invoke Split Miner's concurrency oracle to extract the possible parallelism relations in the log using a randomly chosen parallelism threshold. For each new parallel relation discovered (not present in the current solution), two edges are removed from the DFG, whilst, for each deprecated parallel relation, two edges are added to the DFG.

Fodina PF. Given the current DFG, we analyse its self-loops and short-loops re-

---

[5] By direct suggestion of the main author of Inductive Miner.

Algorithm 13: Generate Neighbours Function (GNF)

```
    input : CurrentDFG $\mathscr{G}_c$, CurrentMarkovianScore $s_c$, Integer $size_n$
1   if getFitnessScore($s_c$) > getPrecisionScore($s_c$) then
2       Set $E_m \leftarrow$ getEdgesForImprovingPrecision($s_c$);
3   else
4       Set $E_m \leftarrow$ getEdgesForImprovingFitness($s_c$);
5   Set $N \leftarrow \varnothing$;
6   while $E_m \neq \varnothing \wedge |N| \neq size_n$ do
7       Edge $e \leftarrow$ getRandomElement($E_m$);
8       NeighbouringDFG $\mathscr{G}_n \leftarrow$ copyDFG($\mathscr{G}_c$);
9       if getFitnessScore($s_c$) > getPrecisionScore($s_c$) then
10          if canRemoveEdge($\mathscr{G}_n$, e) then add $\mathscr{G}_n$ to $N$;
11          ;
12      else
13          addEdge($\mathscr{G}_n$, $e$);
14          add $\mathscr{G}_n$ to $N$;
15  return $N$;
```

lations using random loop thresholds. As a result, a new DFG is generated where a different set of edges is retained as self-loops and short-loops.

Inductive Miner PF. Since Inductive Miner does not perform any manipulation on the DFG, we could not determine an efficient way to perturb the DFG, and we set PF = RF, so that instead of perturbing the current DFG, a new random DFG is generated. This variant of the ILS is called Repetitive Local Search (RLS). In the evaluation (reported in Section 7.3), we use only RLS for Inductive Miner, and both ILS and RLS for Fodina and Split Miner.

To complete the instantiation of our framework, we only need to set the objective function. Given that our goal is the optimization of the accuracy of the APDAs, we chose as objective function the F-score of fitness and precision. Among the existing measures of fitness and precision, we selected the Markovian fitness and precision (defined in Chapter 6), the boolean function variant with order $k = 5$. The rationale for this choice is that these measures of fitness and precision are the fastest to compute among state-of-the-art measures (as shown in Chapter 6). Furthermore, the Markovian fitness (precision) provides a feedback that tells us what edges could be added to (removed from) the DFG to improve the fitness (precision). This feedback allows us to design an effective GNF. In the instantiation of our framework, the objective function's output is a data structure composed of: the Markovian fitness and precision of the model, the F-score, and the mismatches between the model and the event log identified during the computation of the Markovian fitness and precision, i.e. the sets of the edges that could be added (removed) to improve the fitness (precision).

Given this objective function's output, our GNF is described in Algorithm 13. The function receives as input the current DFG ($\mathscr{G}_c$), its objective function score (the data structure s_c), and the number of neighbours to generate ($size_n$). If fitness is greater than precision, we retrieve (from s_c) the set of edges (E_m) that could be removed from $\mathscr{G}_c$ to improve its precision (line 2). Conversely, if precision is greater than fitness, we retrieve (from s_c) the set of edges (E_m) that could be added to $\mathscr{G}_c$ to improve its fitness (line 4). The reasoning behind this design choice is that, given that our objective function is the F-score, it is preferable to increase the lower of the two measures (precision or fitness). i.e. if the fitness is lower, we increase fitness, and conversely if the precision is lower. Once we have E_m, we select randomly one edge from it, we generate a copy of the current DFG ($\mathscr{G}_n$), and we either remove or add the randomly selected edge according to the accuracy measure we want to improve (precision or fitness), see lines 7 to 14. If the removal of an edge generates a disconnected $\mathscr{G}_n$, we do not add this latter to the neighbours set (N), line 11. We keep iterating over E_m until the set is empty (i.e. no mismatching edges are left) or N reaches its max size (i.e. $size_n$). We then return N.

The algorithm ends when the maximum execution time is reached or and the maximum number of iterations it reached (in the experiments below, we set them by default to 5 minutes and 50 iterations).

7.3. Evaluation

7.3.1. Dataset and Experimental Setup

We implemented our optimization framework as a Java command-line application[6] integrating Split Miner, Fodina, and Inductive Miner as the underlying APDAs, and the Markovian accuracy F-score as the objective function (cf. Section 7.2.4).

Using the same public dataset of our benchmark (12 event logs) and eight proprietary logs sourced from several companies in the education, insurance, IT service management, and IP management domains, we discovered from each log 16 models by applying the following techniques: Split Miner with default parameters (SM); Split Miner with hyper-parameter optimization[7](HPO_{sm}); Split Miner optimized with our framework using the following optimization metaheuristics, RLS (RLS_{sm}), ILS (ILS_{sm}), TABU (TABU_{sm}), SIMA (SIMA_{sm}); Fodina with default parameters (FO); Fodina with hyper-parameter optimization[7] (HPO_{fo}); Fodina optimized with our framework using the following optimization metaheuristics, RLS (RLS_{fo}), ILS (ILS_{fo}), TABU (TABU_{fo}), SIMA (SIMA_{fo}); Inductive Miner (IM_{d}); Inductive Miner optimized with our framework using the following optimization metaheuristics, RLS (RLS_{imd}), TABU (TABU_{imd}), SIMA (SIMA_{imd}).

For each of the discovered models we measured accuracy, complexity and discovery time. For the accuracy, we adopted two different sets of measures: one based on *alignments*, computing fitness and precision with the approaches proposed by Adriansyah et al. [158, 181] (alignment-based accuracy); and one based

---

[6] Available under the label "Optimization Framework for Automated Process Discovery" at `http://apromore.org/platform/tools`.

[7] With objective function the Markovian accuracy F-score.

on *Markovian abstractions*, computing fitness and precision with the approaches introduced in Chapter 6 (Markovian accuracy). For assessing the complexity of the models we relied on *size*, Control-Flow Complexity (CFC), and Structuredness.

These measurements allowed us to compare the quality of the models discovered by each baseline APDA (SM, FO, IM_d) against the quality of the models discovered by the respective optimized approaches.

All the experiments were performed on an Intel Core i5-6200U@2.30GHz with 16GB RAM running Windows 10 Pro (64-bit) and JVM 8 with 14GB RAM (10GB Stack and 4GB Heap). The framework implementation, the batch tests, the results, and all the models discovered during the experiments are available for reproducibility purposes at `https://doi.org/10.6084/m9.figshare.7824671.v1`.

7.3.2. Results

Split Miner. Tables 43 and 44 show the results of our comparative evaluation for Split Miner. Each row reports the quality of each discovered process model in terms of accuracy (both alignment-based and Markovian), complexity, and discovery time. We held out from the tables four logs: BPIC13_cp, BPIC13_inc, BPIC17, and PRT9. For these logs, none of the metaheuristics could improve the accuracy of the model already discovered by SM. This is due to the high fitness score achieved by SM in these logs. By design, our metaheuristics try to improve the precision by removing edges, but in these four cases, no edge could be removed without compromising the structure of the model (i.e. the model would become disconnected).

For the remaining 16 logs, all the metaheuristics improved consistently the Markovian F-score w.r.t. SM. Also, all the metaheuristics performed better than HPO_sm, except in two cases (BPIC12 and PRT1). Overall, the most effective optimization metaheuristic was ILS, which delivered the highest Markovian F-score nine times out of 16, followed by SIMA_sm (eight times), RLS_sm and TABU_sm (six times each).

Despite the fact that the objective function of the metaheuristics was the Markovian F-score, all four metaheuristics optimized in half of the cases also the alignment-based F-score. This is due to the fact that any improvement on the Markovian fitness translates into an improvement on the alignment-based fitness, though the same does not hold for the precision. This result highlights the (partial) correlation between the alignment-based and the Markovian accuracies, already reported in the previous chapter. Analysing the complexity of the models, we note that most of the times (nine cases out of 16) the F-score improvement achieved by the metaheuristics comes at the cost of size and CFC. This is expected, since SM tends to discover models with higher precision than fitness (see also Chapter 5). What happens is that to improve the F-score, new behaviour is added to the model in

Event Log	Discovery Approach	Align. Acc.			Markov. Acc. ($k = 5$)			Complexity			Exec. Time(s)
		Fitness	**Prec.**	**F-score**	**Fitness**	**Prec.**	**F-score**	**Size**	**CFC**	**Struct.**	
BPIC12	SM	**0.963**	0.520	0.675	**0.818**	0.139	0.238	51	41	0.69	**3.2**
	HPO$_{\text{sm}}$	0.781	**0.796**	**0.788**	0.575	**0.277**	**0.374**	**40**	**17**	0.58	4295.8
	RLS$_{\text{sm}}$	0.921	0.671	0.776	0.586	0.247	0.348	49	31	**0.90**	159.3
	ILS$_{\text{sm}}$	0.921	0.671	0.776	0.586	0.247	0.348	49	31	**0.90**	159.4
	TABU$_{\text{sm}}$	0.921	0.671	0.776	0.586	0.247	0.348	49	31	**0.90**	140.7
	SIMA$_{\text{sm}}$	0.921	0.671	0.776	0.586	0.247	0.348	49	31	**0.90**	151.1
BPIC14$_{\text{f}}$	SM	0.772	**0.881**	0.823	0.150	**1.000**	0.262	**20**	**14**	**1.00**	**0.8**
	HPO$_{\text{sm}}$	0.852	0.857	0.855	0.449	**1.000**	0.619	22	16	0.59	575.8
	RLS$_{\text{sm}}$	**1.000**	0.771	**0.871**	**1.000**	0.985	**0.992**	28	34	0.54	139.0
	ILS$_{\text{sm}}$	**1.000**	0.771	**0.871**	**1.000**	0.985	**0.992**	28	34	0.54	151.3
	TABU$_{\text{sm}}$	0.955	0.775	0.855	0.856	0.999	0.922	26	31	0.69	154.7
	SIMA$_{\text{sm}}$	**1.000**	0.771	**0.871**	**1.000**	0.985	**0.992**	28	34	0.54	140.3
BPIC15$_{\text{1f}}$	SM	0.899	**0.871**	0.885	0.701	0.726	0.713	**111**	**45**	**0.51**	**0.7**
	HPO$_{\text{sm}}$	**0.962**	0.833	**0.893**	**0.804**	0.670	0.731	117	55	0.45	1242.3
	RLS$_{\text{sm}}$	0.925	0.839	0.880	0.774	0.803	0.788	124	63	0.39	163.6
	ILS$_{\text{sm}}$	0.925	0.839	0.880	0.774	0.803	0.788	124	63	0.39	166.8
	TABU$_{\text{sm}}$	0.948	0.843	0.892	0.774	0.805	**0.789**	125	64	0.33	187.2
	SIMA$_{\text{sm}}$	0.920	0.839	0.878	0.772	**0.807**	**0.789**	125	63	0.43	160.4
BPIC15$_{\text{2f}}$	SM	0.783	**0.877**	0.828	0.514	0.596	0.552	**129**	**49**	**0.36**	**0.6**
	HPO$_{\text{sm}}$	0.808	0.851	0.829	0.561	0.582	0.572	133	56	0.30	1398.9
	RLS$_{\text{sm}}$	0.870	0.797	**0.832**	0.667	0.670	0.668	156	86	0.20	158.3
	ILS$_{\text{sm}}$	0.869	0.795	0.830	0.663	**0.680**	**0.671**	157	86	0.20	157.6
	TABU$_{\text{sm}}$	0.870	0.794	0.830	0.665	0.667	0.666	150	83	0.23	176.8
	SIMA$_{\text{sm}}$	**0.871**	0.775	0.820	**0.677**	0.662	0.669	159	93	0.26	167.4
BPIC15$_{\text{3f}}$	SM	0.774	**0.925**	0.843	0.436	0.764	0.555	**96**	**35**	0.49	**0.5**
	HPO$_{\text{sm}}$	0.783	0.910	0.842	0.477	0.691	0.564	99	39	**0.56**	9230.4
	RLS$_{\text{sm}}$	0.812	0.903	**0.855**	0.504	**0.775**	0.611	110	53	0.35	151.5
	ILS$_{\text{sm}}$	**0.833**	0.868	0.850	0.533	**0.775**	**0.631**	120	66	0.23	153.8
	TABU$_{\text{sm}}$	0.832	0.852	0.842	0.558	0.690	0.617	121	64	0.23	173.4
	SIMA$_{\text{sm}}$	0.827	0.839	0.833	**0.565**	0.694	0.623	123	71	0.18	159.4
BPIC15$_{\text{4f}}$	SM	0.762	**0.886**	0.820	0.516	0.615	0.562	**101**	**37**	**0.27**	**0.5**
	HPO$_{\text{sm}}$	0.785	0.860	0.821	0.558	0.578	0.568	103	40	**0.27**	736.4
	RLS$_{\text{sm}}$	0.825	0.854	**0.839**	0.634	**0.672**	0.652	114	57	0.21	146.9
	ILS$_{\text{sm}}$	**0.853**	0.807	0.829	**0.649**	0.657	**0.653**	117	64	**0.27**	147.8
	TABU$_{\text{sm}}$	0.811	0.794	0.803	0.642	0.661	0.651	115	61	0.24	161.7
	SIMA$_{\text{sm}}$	0.847	0.812	0.829	0.624	0.649	0.636	117	61	0.18	148.2
BPIC15$_{\text{5f}}$	SM	0.806	0.915	0.857	0.555	0.598	0.576	110	38	**0.34**	**0.6**
	HPO$_{\text{sm}}$	0.789	**0.941**	**0.858**	0.529	0.655	0.585	**102**	**30**	0.33	972.3
	RLS$_{\text{sm}}$	0.868	0.813	0.840	0.737	0.731	0.734	137	78	0.14	159.3
	ILS$_{\text{sm}}$	0.868	0.813	0.840	0.737	0.731	0.734	137	78	0.14	153.8
	TABU$_{\text{sm}}$	**0.885**	0.818	0.850	**0.739**	**0.746**	**0.743**	137	79	0.14	173.3
	SIMA$_{\text{sm}}$	0.867	0.811	0.838	0.734	0.727	0.731	137	78	0.16	154.3
RTFMP	SM	**0.996**	0.958	0.977	**0.959**	0.311	0.470	22	17	**0.46**	**2.9**
	HPO$_{\text{sm}}$	0.887	**1.000**	0.940	0.685	0.696	0.690	**20**	**9**	0.35	2452.7
	RLS$_{\text{sm}}$	0.988	**1.000**	**0.994**	0.899	0.794	0.843	22	14	**0.46**	142.8
	ILS$_{\text{sm}}$	0.988	**1.000**	**0.994**	0.899	0.794	0.843	22	14	**0.46**	143.8
	TABU$_{\text{sm}}$	0.988	**1.000**	**0.994**	0.899	0.794	0.843	22	14	**0.46**	114.8
	SIMA$_{\text{sm}}$	0.986	**1.000**	0.993	0.875	**0.893**	**0.884**	23	15	0.39	131.0
SEPSIS	SM	0.764	**0.706**	**0.734**	0.349	**0.484**	0.406	**32**	**23**	**0.94**	**0.4**
	HPO$_{\text{sm}}$	**0.925**	0.588	0.719	**0.755**	0.293	0.423	33	34	0.39	28,846
	RLS$_{\text{sm}}$	0.839	0.630	0.720	0.508	0.430	**0.466**	35	29	0.77	145.4
	ILS$_{\text{sm}}$	0.812	0.625	0.706	0.455	0.436	0.445	35	28	0.86	157.1
	TABU$_{\text{sm}}$	0.839	0.630	0.720	0.508	0.430	**0.466**	35	29	0.77	137.0
	SIMA$_{\text{sm}}$	0.806	0.613	0.696	0.477	0.445	0.460	35	30	0.77	137.2

Table 43: Comparative evaluation results for the public logs - Split Miner.

Event Log	**Discovery Method**	**Align. Acc.**			**Markov. Acc.** ($k = 5$)			**Complexity**			**Exec. Time(s)**
		Fitness	**Prec.**	**F-score**	**Fitness**	**Prec.**	**F-score**	**Size**	**CFC**	**Struct.**	
PRT1	SM	0.976	**0.974**	**0.975**	0.730	0.669	0.698	20	**14**	**1.00**	**0.4**
	HPO$_{\text{sm}}$	**0.999**	0.948	0.972	**0.989**	0.620	0.762	**19**	**14**	0.53	298.3
	RLS$_{\text{sm}}$	0.976	**0.974**	**0.975**	0.730	0.669	0.698	20	**14**	**1.00**	155.3
	ILS$_{\text{sm}}$	0.976	**0.974**	**0.975**	0.730	0.669	0.698	20	**14**	**1.00**	153.2
	TABU$_{\text{sm}}$	0.976	**0.974**	**0.975**	0.730	0.669	0.698	20	**14**	**1.00**	10.3
	SIMA$_{\text{sm}}$	0.983	0.964	0.974	0.814	**0.722**	**0.765**	20	15	**1.00**	132.6
PRT2	SM	0.795	0.581	0.671	0.457	**0.913**	0.609	29	23	**1.00**	**0.3**
	HPO$_{\text{sm}}$	0.826	**0.675**	**0.743**	0.501	0.830	0.625	**21**	**13**	0.67	406.4
	RLS$_{\text{sm}}$	0.886	0.421	0.571	0.629	0.751	0.685	29	34	**1.00**	141.4
	ILS$_{\text{sm}}$	**0.890**	0.405	0.557	**0.645**	0.736	**0.688**	29	35	**1.00**	172.3
	TABU$_{\text{sm}}$	0.866	0.425	0.570	0.600	0.782	0.679	29	33	**1.00**	143.1
	SIMA$_{\text{sm}}$	0.886	0.424	0.574	0.629	0.751	0.685	29	34	**1.00**	139.7
PRT3	SM	0.882	0.887	0.885	0.381	0.189	0.252	31	23	0.58	**0.4**
	HPO$_{\text{sm}}$	0.890	0.899	0.895	0.461	0.518	0.488	**26**	**14**	**0.81**	290.2
	RLS$_{\text{sm}}$	**0.945**	**0.902**	**0.923**	**0.591**	0.517	0.551	31	23	0.55	138.4
	ILS$_{\text{sm}}$	**0.945**	**0.902**	**0.923**	**0.591**	0.517	0.551	31	23	0.55	144.2
	TABU$_{\text{sm}}$	0.944	**0.902**	0.922	0.589	**0.519**	**0.552**	30	20	0.60	134.7
	SIMA$_{\text{sm}}$	**0.945**	**0.902**	**0.923**	**0.591**	0.517	0.551	31	23	0.55	133.7
PRT4	SM	0.884	**1.000**	0.938	0.483	**1.000**	0.652	**25**	**15**	**0.96**	**0.5**
	HPO$_{\text{sm}}$	0.973	0.930	**0.951**	0.929	0.989	0.958	26	24	0.31	867.5
	RLS$_{\text{sm}}$	**0.997**	0.903	0.948	**0.993**	0.990	**0.992**	26	28	0.92	140.1
	ILS$_{\text{sm}}$	**0.997**	0.903	0.948	**0.993**	0.990	**0.992**	26	28	0.92	152.3
	TABU$_{\text{sm}}$	0.955	0.914	0.934	0.883	0.988	0.932	26	26	0.77	138.6
	SIMA$_{\text{sm}}$	**0.997**	0.903	0.948	**0.993**	0.990	**0.992**	26	28	0.92	136.9
PRT6	SM	0.937	**1.000**	**0.967**	0.542	**1.000**	0.703	**15**	**4**	**1.00**	**0.3**
	HPO$_{\text{sm}}$	0.937	**1.000**	**0.967**	0.542	**1.000**	0.703	**15**	**4**	**1.00**	105.1
	RLS$_{\text{sm}}$	**0.984**	0.928	0.955	**0.840**	0.818	**0.829**	22	14	0.41	141.1
	ILS$_{\text{sm}}$	**0.984**	0.928	0.955	**0.840**	0.818	**0.829**	22	14	0.41	144.2
	TABU$_{\text{sm}}$	**0.984**	0.928	0.955	**0.840**	0.818	**0.829**	22	14	0.41	124.9
	SIMA$_{\text{sm}}$	**0.984**	0.928	0.955	**0.840**	0.818	**0.829**	22	14	0.41	131.2
PRT7	SM	0.914	0.999	0.954	0.650	**1.000**	0.788	29	10	0.48	**0.6**
	HPO$_{\text{sm}}$	0.944	**1.000**	0.971	0.772	**1.000**	0.871	**22**	**9**	0.64	173.1
	RLS$_{\text{sm}}$	**0.993**	**1.000**	**0.996**	**0.933**	**1.000**	**0.965**	23	11	**0.78**	139.2
	ILS$_{\text{sm}}$	**0.993**	**1.000**	**0.996**	**0.933**	**1.000**	**0.965**	23	11	**0.78**	142.9
	TABU$_{\text{sm}}$	**0.993**	**1.000**	**0.996**	**0.933**	**1.000**	**0.965**	23	11	**0.78**	134.0
	SIMA$_{\text{sm}}$	**0.993**	**1.000**	**0.996**	**0.933**	**1.000**	**0.965**	23	11	**0.78**	131.9
PRT10	SM	**0.970**	0.943	**0.956**	**0.905**	0.206	0.335	45	47	0.84	**0.5**
	HPO$_{\text{sm}}$	0.936	0.943	0.939	0.810	0.243	0.374	**30**	**22**	0.73	1214.3
	RLS$_{\text{sm}}$	0.917	**0.989**	0.952	0.741	**0.305**	**0.432**	44	41	**0.86**	153.0
	ILS$_{\text{sm}}$	0.917	**0.989**	0.952	0.741	**0.305**	**0.432**	44	41	**0.86**	155.4
	TABU$_{\text{sm}}$	0.917	**0.989**	0.952	0.741	**0.305**	**0.432**	44	41	**0.86**	117.6
	SIMA$_{\text{sm}}$	0.917	**0.989**	0.952	0.741	**0.305**	**0.432**	44	41	**0.86**	136.7

Table 44: Comparative evaluation results for the proprietary logs - Split Miner.

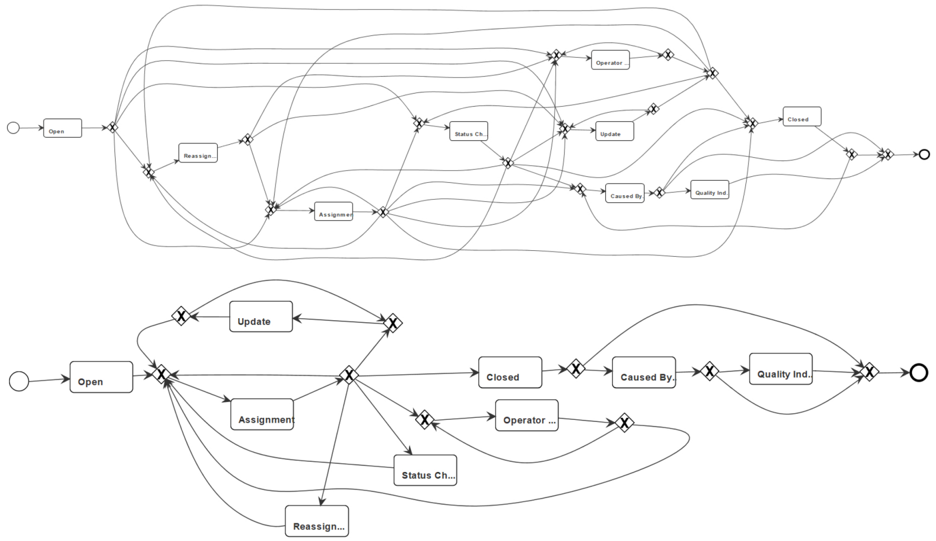


Figure 29: BPIC14$_\text{f}$ models discovered with SIMA$_\text{sm}$ (above) and SM (below).

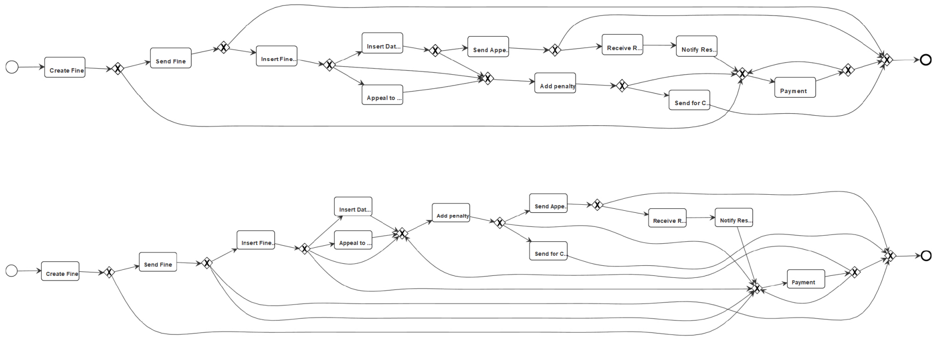


Figure 30: RTFMP models discovered with SIMA$_\text{sm}$ (above) and SM (below).

the form of new edges (note that new nodes are never added). Adding new edges leads to new gateways and consequently to increasing size and CFC. On the other hand, when the precision is lower than fitness and the metaheuristic aims to increase the value of this measure to improve the overall F-score, the result is the opposite: the model complexity reduces as edges are removed. This is the case of the RTFMP and PRT10 logs. As an example of the two possible scenarios, Figure 29 shows the models discovered by SIMA$_\text{sm}$ and SM from the BPIC14$_\text{f}$ log (where the model discovered by SIMA$_\text{sm}$ is more complex than that obtained with SM), while Figure 30 shows the models discovered by SIMA$_\text{sm}$ and SM from the RTFMP log (where the model discovered by SIMA$_\text{sm}$ is simpler). Comparing the results obtained by the metaheuristics with HPO$_\text{sm}$, we can see that our approach allows us to discover models that cannot be discovered simply by tuning the hyperparameters of SM. This relates to the solution space exploration. Indeed, while

HPO_{sm} can only explore a limited number of solutions (DFGs), i.e. those that can be generated by the underlying APDA, SM in this case, by varying its hyperparameters, the metaheuristics go beyond the solution space of HPO_{sm} by exploring new DFGs in a pseudo-random manner.

In terms of execution times, the four metaheuristics perform similarly, having an average discovery time close to 150 seconds. While this is considerably higher than the execution time of SM ($\sim$ 1 second on average), it is much lower than HPO_{sm}, while consistently achieving higher accuracy.

Fodina. Tables 45 and 46 report the results of our comparative evaluation for Fodina. In these tables, we used "-" to report that a given accuracy measurement could not be reliably obtained due to the unsoundness of the discovered process model. We held out from the tables two logs: BPIC12 and SEPSIS, because none of the 6 approaches was able to discover a sound process model, this is due to Fodina design, which does not guarantee any soundness.

Considering the remaining 18 logs, 11 times *all* the metaheuristics improved the Markovian F-score w.r.t. to HPO_{fo} (and consequently FO), whilst 16 times at least one metaheuristic outperformed both FO and HPO_{fo}. The only two cases where none of the metaheuristics was able to discover a more accurate process model than HPO_{fo} are PRT2 and BPIC14_{f}. In the latter log, because all the metaheuristics discovered the same model of HPO_{fo}. In the former log, because none of the metaheuristics discovered a sound process model within the given timeout of five minutes, however, we note that HPO_{fo} took almost four hours to discover a sound process model from the PRT2 log. Among the optimization metaheuristics, the one performing the best was TABU_{fo}, which achieved 14 times out of 18 the best Markovian F-score, followed by ILS_{fo} (10 times).

In this case, the results achieved by the metaheuristics on the alignment-based F-score are more remarkable than the case of SM, and in-line with the results obtained on the Markovian F-score. Indeed, 50% of the times, *all* the metaheuristics were able to outperform both FO and HPO_{fo} on the alignment-based F-score, and more than 80% of the times, at least one metaheuristic scored higher alignment-based F-score than FO and HPO_{fo}. Such a result is impressive considering that the objective function of the metaheuristics was the Markovian F-score.

Regarding the complexity of the models discovered by the metaheuristics, more than 50% of the times, it is lower than the complexity of the models discovered by FO and HPO_{fo}, and in the remaining cases in-line with the two baselines. Such a difference with the results we obtained for SM relates to the following two factors: (i) SM discovers much simpler models than FO, and any further improvement is difficult to achieve; (ii) FO natively discovers more fitting models than SM, so that the metaheuristics aim at improving the precision of the discovered models ultimately removing model's edges and reducing its complexity.

In terms of execution times, the four metaheuristics perform similarly, with an execution time between 150 and 300 seconds, slightly higher than the case of SM.

Event	**Discovery**	**Align. Acc.**			**Markov. Acc.** ($k = 5$)			**Complexity**			**Exec.**
Log	**Approach**	**Fitness**	**Prec.**	**F-score**	**Fitness**	**Prec.**	**F-score**	**Size**	**CFC**	**Struct.**	**Time(s)**
BPIC13$_{\text{cp}}$	FO	**0.999**	0.879	0.935	**0.997**	0.647	0.784	13	10	0.77	**0.1**
	HPO$_{\text{fo}}$	**0.999**	0.879	0.935	**0.997**	0.647	0.784	13	10	0.77	17.7
	RLS$_{\text{fo}}$	0.994	**0.963**	**0.978**	0.947	**0.864**	**0.904**	**12**	9	0.67	290.6
	ILS$_{\text{fo}}$	0.994	0.880	0.934	0.935	0.758	0.837	**12**	**8**	**0.92**	151.2
	TABU$_{\text{fo}}$	0.994	**0.963**	**0.978**	0.947	**0.864**	**0.904**	**12**	9	0.67	95.2
	SIMA$_{\text{fo}}$	0.994	0.880	0.934	0.935	0.758	0.837	**12**	**8**	**0.92**	130.0
BPIC13$_{\text{inc}}$	FO	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	**0.291**
	HPO$_{\text{fo}}$	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	112.0
	RLS$_{\text{fo}}$	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	304.7
	ILS$_{\text{fo}}$	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	180.1
	TABU$_{\text{fo}}$	**0.998**	0.743	0.852	**0.987**	**0.604**	**0.749**	14	15	**1.00**	129.0
	SIMA$_{\text{fo}}$	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	146.1
BPIC14$_{\text{f}}$	FO	-	-	-	-	-	-	37	46	0.41	**36.8**
	HPO$_{\text{fo}}$	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	8612.7
	RLS$_{\text{fo}}$	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	370.7
	ILS$_{\text{fo}}$	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	365.5
	TABU$_{\text{fo}}$	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	358.5
	SIMA$_{\text{fo}}$	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	300.2
BPIC15$_{\text{1f}}$	FO	**1.000**	0.760	0.860	**1.000**	0.480	0.650	146	91	0.26	**0.3**
	HPO$_{\text{fo}}$	**1.000**	0.756	0.861	**1.000**	0.479	0.648	146	91	0.26	130.5
	RLS$_{\text{fo}}$	0.916	0.829	0.870	0.804	0.772	0.788	131	69	0.24	301.9
	ILS$_{\text{fo}}$	0.916	0.829	0.870	0.804	0.772	0.788	131	69	0.24	198.4
	TABU$_{\text{fo}}$	0.916	0.830	0.871	0.802	0.778	**0.790**	129	**67**	0.33	177.5
	SIMA$_{\text{fo}}$	0.918	**0.833**	**0.873**	0.777	**0.799**	0.788	**127**	**67**	**0.34**	174.4
BPIC15$_{\text{2f}}$	FO	-	-	-	-	-	-	195	159	0.09	**48.5**
	HPO$_{\text{fo}}$	-	-	-	-	-	-	187	145	0.11	118.7
	RLS$_{\text{fo}}$	-	-	-	-	-	-	181	131	0.09	306.0
	ILS$_{\text{fo}}$	-	-	-	-	-	-	**175**	**120**	0.11	276.1
	TABU$_{\text{fo}}$	**0.876**	**0.754**	**0.810**	**0.653**	**0.608**	**0.630**	177	**120**	**0.13**	262.3
	SIMA$_{\text{fo}}$	-	-	-	-	-	-	**175**	121	0.12	284.1
BPIC15$_{\text{3f}}$	FO	-	-	-	-	-	-	174	164	0.06	**4.3**
	HPO$_{\text{fo}}$	**0.983**	0.601	0.746	**0.925**	0.208	0.339	163	161	0.07	402.9
	RLS$_{\text{fo}}$	-	-	-	-	-	-	166	141	0.07	303.5
	ILS$_{\text{fo}}$	0.924	**0.713**	**0.805**	0.701	**0.444**	**0.543**	**158**	**131**	**0.10**	247.1
	TABU$_{\text{fo}}$	-	-	-	-	-	-	163	**131**	0.09	235.5
	SIMA$_{\text{fo}}$	-	-	-	-	-	-	163	**131**	0.09	241.8
BPIC15$_{\text{4f}}$	FO	-	-	-	-	-	-	157	127	0.15	**1.3**
	HPO$_{\text{fo}}$	**0.995**	0.660	0.793	**0.973**	0.302	0.461	153	126	0.14	443.0
	RLS$_{\text{fo}}$	0.887	0.790	0.836	0.708	0.610	0.655	**127**	77	**0.17**	308.3
	ILS$_{\text{fo}}$	0.882	0.801	**0.839**	0.697	0.628	0.661	**127**	75	**0.17**	300.5
	TABU$_{\text{fo}}$	0.864	**0.806**	0.834	0.675	**0.652**	**0.663**	**127**	**74**	**0.17**	274.5
	SIMA$_{\text{fo}}$	0.882	0.801	**0.839**	0.697	0.628	0.661	**127**	75	**0.17**	252.2
BPIC15$_{\text{5f}}$	FO	**1.000**	0.698	0.822	**1.000**	0.362	0.532	166	125	**0.15**	**2.4**
	HPO$_{\text{fo}}$	**1.000**	0.698	0.822	**1.000**	0.362	0.532	166	125	**0.15**	238.1
	RLS$_{\text{fo}}$	0.886	0.810	0.846	0.727	0.703	0.715	150	94	0.11	303.4
	ILS$_{\text{fo}}$	0.884	**0.819**	**0.850**	0.719	0.724	0.722	147	90	0.13	268.1
	TABU$_{\text{fo}}$	0.886	0.814	0.849	0.723	0.730	0.727	149	92	0.11	217.1
	SIMA$_{\text{fo}}$	0.884	0.808	0.844	0.721	**0.743**	**0.732**	**141**	**83**	0.14	208.5
BPIC17$_{\text{f}}$	FO	**1.000**	**0.675**	**0.806**	**1.000**	0.330	0.496	35	22	0.69	**22.4**
	HPO$_{\text{fo}}$	**1.000**	**0.675**	**0.806**	**1.000**	0.330	0.496	35	22	**0.71**	9755.7
	RLS$_{\text{fo}}$	0.999	**0.675**	**0.806**	0.997	**0.331**	**0.497**	**33**	**20**	0.70	309.9
	ILS$_{\text{fo}}$	0.999	**0.675**	**0.806**	0.997	**0.331**	**0.497**	**33**	**20**	0.70	313.5
	TABU$_{\text{fo}}$	0.999	**0.675**	**0.806**	0.997	**0.331**	**0.497**	**33**	**20**	0.70	305.7
	SIMA$_{\text{fo}}$	0.999	**0.675**	**0.806**	0.997	**0.331**	**0.497**	**33**	**20**	0.70	319.2
RTFMP	FO	**0.996**	0.933	0.964	**0.937**	0.148	0.256	31	32	0.19	**0.4**
	HPO$_{\text{fo}}$	0.884	**1.000**	0.939	0.646	0.857	0.737	**18**	**7**	**0.56**	2666.2
	RLS$_{\text{fo}}$	0.987	**1.000**	**0.994**	0.848	**0.938**	**0.890**	26	25	0.12	268.7
	ILS$_{\text{fo}}$	0.987	**1.000**	**0.994**	0.848	**0.938**	**0.890**	26	25	0.12	134.1
	TABU$_{\text{fo}}$	0.987	**1.000**	**0.994**	0.848	**0.938**	**0.890**	26	25	0.12	131.2
	SIMA$_{\text{fo}}$	0.987	**1.000**	0.993	0.847	0.923	0.883	28	27	0.11	133.9

Table 45: Comparative evaluation results for the public logs - Fodina.

Event Log	Discovery Method	Align. Acc.			Markov. Acc. ($k = 5$)			Complexity			Exec. Time(s)
		Fitness	Prec.	F-score	Fitness	Prec.	F-score	Size	CFC	Struct.	
PRT1	FO	-	-	-	-	-	-	30	28	0.53	**0.2**
	HPO$_{\text{fo}}$	0.998	0.925	0.960	**0.988**	0.739	0.845	**21**	17	0.81	402.6
	RLS$_{\text{fo}}$	0.988	**0.964**	**0.976**	0.888	0.827	0.857	**21**	**16**	**0.86**	302.7
	ILS$_{\text{fo}}$	0.988	**0.964**	**0.976**	0.888	0.827	0.857	**21**	**16**	**0.86**	183.1
	TABU$_{\text{fo}}$	**0.994**	0.957	**0.976**	0.981	**0.844**	**0.907**	**21**	17	**0.86**	149.3
	SIMA$_{\text{fo}}$	0.988	**0.964**	**0.976**	0.888	0.827	0.857	**21**	**16**	**0.86**	154.0
PRT2	FO	-	-	-	-	-	-	38	**45**	0.76	**92.7**
	HPO$_{\text{fo}}$	**1.000**	**0.276**	**0.432**	**0.998**	**0.148**	**0.258**	**29**	78	**1.00**	12937.1
	RLS$_{\text{fo}}$	-	-	-	-	-	-	48	56	0.08	301.0
	ILS$_{\text{fo}}$	-	-	-	-	-	-	48	56	0.08	308.1
	TABU$_{\text{fo}}$	-	-	-	-	-	-	53	70	0.08	313.0
	SIMA$_{\text{fo}}$	-	-	-	-	-	-	-	-	-	854.9
PRT3	FO	**0.999**	0.847	0.917	**0.993**	0.269	0.423	**34**	**37**	0.32	**0.2**
	HPO$_{\text{fo}}$	-	-	-	-	-	-	73	93	0.18	756.5
	RLS$_{\text{fo}}$	0.963	**0.902**	**0.932**	0.679	**0.446**	**0.539**	37	38	**0.35**	306.6
	ILS$_{\text{fo}}$	0.963	**0.902**	**0.932**	0.679	**0.446**	**0.539**	37	38	**0.35**	157.3
	TABU$_{\text{fo}}$	0.963	**0.902**	**0.932**	0.679	**0.446**	**0.539**	37	38	**0.35**	138.1
	SIMA$_{\text{fo}}$	0.963	**0.902**	**0.932**	0.679	**0.446**	**0.539**	37	38	**0.35**	143.0
PRT4	FO	-	-	-	-	-	-	37	40	0.54	**46.0**
	HPO$_{\text{fo}}$	**1.000**	0.858	0.924	**1.000**	0.965	0.982	32	41	0.50	10914.5
	RLS$_{\text{fo}}$	0.997	0.859	0.923	0.993	0.990	0.991	31	37	0.52	317.4
	ILS$_{\text{fo}}$	0.997	**0.903**	**0.948**	0.993	**0.993**	**0.993**	**27**	**32**	**0.74**	314.4
	TABU$_{\text{fo}}$	0.997	**0.903**	**0.948**	0.993	**0.993**	**0.993**	**27**	**32**	**0.74**	300.1
	SIMA$_{\text{fo}}$	0.977	0.887	0.930	0.793	0.963	0.870	32	38	0.50	309.1
PRT6	FO	**1.000**	0.908	0.952	**1.000**	0.632	0.775	**22**	17	**0.41**	**0.1**
	HPO$_{\text{fo}}$	**1.000**	0.908	0.952	**1.000**	0.632	0.775	**22**	17	**0.41**	25.0
	RLS$_{\text{fo}}$	0.984	**0.928**	**0.955**	0.840	**0.818**	**0.829**	**22**	**14**	**0.41**	278.8
	ILS$_{\text{fo}}$	0.984	**0.928**	**0.955**	0.840	**0.818**	**0.829**	**22**	**14**	**0.41**	140.0
	TABU$_{\text{fo}}$	0.984	**0.928**	**0.955**	0.840	**0.818**	**0.829**	**22**	**14**	**0.41**	129.3
	SIMA$_{\text{fo}}$	0.984	**0.928**	**0.955**	0.840	**0.818**	**0.829**	**22**	**14**	**0.41**	131.2
PRT7	FO	0.990	**1.000**	0.995	0.906	**1.000**	0.951	**26**	**16**	**0.39**	**0.3**
	HPO$_{\text{fo}}$	0.990	**1.000**	0.995	0.906	**1.000**	0.951	**26**	**16**	**0.39**	50.2
	RLS$_{\text{fo}}$	**0.993**	**1.000**	**0.997**	**0.933**	**1.000**	**0.966**	28	22	0.36	287.6
	ILS$_{\text{fo}}$	**0.993**	**1.000**	**0.997**	**0.933**	**1.000**	**0.966**	28	22	0.36	140.3
	TABU$_{\text{fo}}$	**0.993**	**1.000**	**0.997**	**0.933**	**1.000**	**0.966**	28	22	0.36	129.7
	SIMA$_{\text{fo}}$	**0.993**	**1.000**	**0.997**	**0.933**	**1.000**	**0.966**	28	22	0.36	132.1
PRT9	FO	-	-	-	-	-	-	32	45	0.72	**53.1**
	HPO$_{\text{fo}}$	-	-	-	-	-	-	24	18	0.54	2799.5
	RLS$_{\text{fo}}$	**0.969**	0.999	**0.984**	**0.894**	0.893	0.893	23	21	**0.91**	301.5
	ILS$_{\text{fo}}$	-	-	-	-	-	-	34	26	0.15	263.2
	TABU$_{\text{fo}}$	-	-	-	-	-	-	36	30	0.14	185.8
	SIMA$_{\text{fo}}$	0.968	**1.000**	**0.984**	0.887	**0.956**	**0.920**	**20**	**17**	0.80	278.4
PRT10	FO	**0.990**	0.922	0.955	**0.961**	0.087	0.159	52	85	**0.64**	**0.2**
	HPO$_{\text{fo}}$	0.872	0.958	0.913	0.659	0.786	0.717	**35**	**28**	0.60	750.8
	RLS$_{\text{fo}}$	0.964	**0.965**	**0.965**	0.870	**0.813**	**0.840**	44	46	0.25	301.1
	ILS$_{\text{fo}}$	0.964	**0.965**	**0.965**	0.870	**0.813**	**0.840**	44	46	0.25	195.0
	TABU$_{\text{fo}}$	0.964	**0.965**	**0.965**	0.870	**0.813**	**0.840**	44	46	0.25	165.0
	SIMA$_{\text{fo}}$	0.965	0.963	0.964	0.874	0.809	**0.840**	44	47	0.25	161.2

Table 46: Comparative evaluation results for the proprietary logs - Fodina.

Event Log	**Discovery Approach**	**Align. Acc.**			**Markov. Acc. ($k = 5$)**			**Complexity**			**Exec. Time(s)**
		Fitness	**Prec.**	**F-score**	**Fitness**	**Prec.**	**F-score**	**Size**	**CFC**	**Struct.**	
BPIC12	IM$_{d}$	**1.000**	0.168	0.287	**1.000**	<0.001	<0.001	**30**	28	1.00	**0.7**
	RLS$_{imd}$	0.661	0.763	0.708	0.220	0.163	0.187	40	21	1.00	300.0
	TABU$_{imd}$	0.661	0.763	0.708	0.220	0.163	0.187	40	21	1.00	309.4
	SIMA$_{imd}$	0.660	**0.805**	**0.725**	0.204	**0.223**	**0.213**	39	**19**	1.00	308.7
BPIC13$_{cp}$	IM$_{d}$	**1.000**	0.862	0.926	**0.999**	0.161	0.277	15	11	1.00	**0.4**
	RLS$_{imd}$	0.984	**0.889**	0.934	0.882	**0.424**	0.573	**9**	**5**	1.00	301.4
	TABU$_{imd}$	0.990	0.888	**0.936**	0.942	0.414	**0.575**	10	7	1.00	101.8
	SIMA$_{imd}$	0.984	**0.889**	0.934	0.882	**0.424**	0.573	**9**	**5**	1.00	300.5
BPIC13$_{inc}$	IM$_{d}$	**1.000**	0.673	0.805	**1.000**	0.109	0.197	**10**	9	1.00	**0.5**
	RLS$_{imd}$	0.895	**0.921**	**0.908**	0.679	**0.517**	**0.587**	**10**	**6**	1.00	301.5
	TABU$_{imd}$	0.895	**0.921**	**0.908**	0.679	**0.517**	**0.587**	**10**	**6**	1.00	71.9
	SIMA$_{imd}$	0.895	**0.921**	**0.908**	0.679	**0.517**	**0.587**	**10**	**6**	1.00	300.7
BPIC14$_{f}$	IM$_{d}$	0.861	**0.782**	**0.820**	0.507	**0.814**	0.625	27	16	1.00	**0.8**
	RLS$_{imd}$	**0.977**	0.676	0.799	**0.918**	0.447	0.601	**16**	**11**	1.00	302.5
	TABU$_{imd}$	0.949	0.673	0.788	0.859	0.505	**0.636**	17	13	1.00	303.3
	SIMA$_{imd}$	**0.977**	0.676	0.799	**0.918**	0.447	0.601	**16**	**11**	1.00	300.8
BPIC17$_{f}$	IM$_{d}$	**1.000**	0.679	**0.808**	**1.000**	**0.284**	**0.442**	34	23	1.00	**1.5**
	RLS$_{imd}$	0.674	0.815	0.738	0.241	0.214	0.227	**27**	**11**	1.00	302.7
	TABU$_{imd}$	0.693	**0.817**	0.750	0.262	0.204	0.230	28	13	1.00	83.8
	SIMA$_{imd}$	0.674	0.815	0.738	0.241	0.214	0.227	**27**	**11**	1.00	301.0
RTFMP	IM$_{d}$	**1.000**	0.543	0.704	**1.000**	0.003	0.005	**15**	12	1.00	**0.8**
	RLS$_{imd}$	0.938	0.886	0.911	0.784	0.379	0.511	21	14	1.00	321.1
	TABU$_{imd}$	0.938	0.886	0.911	0.784	0.379	0.511	21	14	1.00	52.1
	SIMA$_{imd}$	0.917	**0.907**	**0.912**	0.780	**0.625**	**0.694**	19	**9**	1.00	300.8
SEPSIS	IM$_{d}$	**1.000**	0.291	0.451	**0.918**	0.006	0.012	**24**	23	1.00	**0.4**
	RLS$_{imd}$	0.796	**0.684**	**0.736**	0.367	**0.363**	0.365	27	18	1.00	305.5
	TABU$_{imd}$	0.796	**0.684**	**0.736**	0.367	**0.363**	0.365	27	18	1.00	306.9
	SIMA$_{imd}$	0.813	0.581	0.678	0.482	0.310	**0.377**	25	**16**	1.00	301.6
PRT1	IM$_{d}$	**1.000**	0.748	0.856	**1.000**	0.025	0.048	**14**	11	1.00	**0.5**
	RLS$_{imd}$	0.974	**0.946**	**0.960**	0.692	**0.707**	**0.699**	16	**10**	1.00	304.4
	TABU$_{imd}$	0.971	**0.946**	0.958	0.692	**0.707**	**0.699**	17	**10**	1.00	304.0
	SIMA$_{imd}$	0.974	**0.946**	**0.960**	0.692	**0.707**	**0.699**	16	**10**	1.00	300.7
PRT2	IM$_{d}$	**1.000**	0.243	0.390	**1.000**	0.109	0.196	**13**	**11**	1.00	**0.9**
	RLS$_{imd}$	0.811	**0.464**	**0.591**	0.588	0.601	**0.594**	18	13	1.00	305.5
	TABU$_{imd}$	0.788	0.461	0.581	0.542	0.566	0.554	16	**11**	1.00	303.0
	SIMA$_{imd}$	0.792	0.413	0.543	0.524	**0.674**	0.590	18	13	1.00	307.3
PRT3	IM$_{d}$	0.827	0.890	0.857	0.328	0.253	0.286	**26**	**10**	1.00	**0.4**
	RLS$_{imd}$	0.914	0.896	0.905	0.501	**0.593**	0.543	**26**	14	1.00	305.1
	TABU$_{imd}$	**0.933**	**0.900**	**0.917**	**0.626**	0.592	**0.608**	28	15	1.00	302.6
	SIMA$_{imd}$	0.930	0.898	0.914	0.562	0.539	0.550	29	17	1.00	300.8
PRT4	IM$_{d}$	0.880	0.811	0.844	0.876	**0.967**	0.919	27	**13**	1.00	**0.5**
	RLS$_{imd}$	**0.962**	**0.879**	**0.919**	**1.000**	0.956	**0.977**	**19**	**13**	1.00	301.0
	TABU$_{imd}$	**0.962**	**0.879**	**0.919**	**1.000**	0.956	**0.977**	**19**	**13**	1.00	307.2
	SIMA$_{imd}$	**0.962**	**0.879**	**0.919**	**1.000**	0.956	**0.977**	**19**	**13**	1.00	300.8
PRT6	IM$_{d}$	0.917	**0.988**	0.951	0.524	0.350	0.420	18	**6**	1.00	**0.4**
	RLS$_{imd}$	**0.953**	0.987	**0.969**	**0.674**	**0.941**	**0.785**	**17**	7	1.00	304.2
	TABU$_{imd}$	0.905	0.915	0.910	0.488	0.903	0.634	18	10	1.00	643.0
	SIMA$_{imd}$	**0.953**	0.987	**0.969**	**0.674**	**0.941**	**0.785**	**17**	7	1.00	300.6
PRT7	IM$_{d}$	0.852	0.997	0.919	0.618	0.407	0.491	21	**5**	1.00	**0.4**
	RLS$_{imd}$	0.917	**1.000**	0.957	**0.700**	**1.000**	**0.824**	**20**	6	1.00	305.5
	TABU$_{imd}$	0.917	**1.000**	0.957	**0.700**	**1.000**	**0.824**	**20**	6	1.00	1013.9
	SIMA$_{imd}$	**0.960**	0.988	**0.974**	0.664	0.752	0.705	23	13	1.00	309.6
PRT9	IM$_{d}$	0.586	0.461	0.516	0.078	0.014	0.024	22	15	1.00	**2.6**
	RLS$_{imd}$	0.945	**1.000**	0.972	0.851	**1.000**	**0.919**	16	9	1.00	304.2
	TABU$_{imd}$	0.946	**1.000**	0.972	0.856	0.947	0.899	18	10	1.00	306.6
	SIMA$_{imd}$	**0.954**	**1.000**	**0.976**	**0.890**	0.909	0.899	**15**	**8**	1.00	300.0
PRT10	IM$_{d}$	0.530	0.656	0.586	0.386	0.000	0.001	36	28	1.00	**0.5**
	RLS$_{imd}$	0.859	**0.961**	0.907	0.664	0.691	0.677	**30**	**24**	1.00	300.3
	TABU$_{imd}$	**0.912**	0.907	**0.909**	**0.790**	0.484	0.600	**30**	**24**	1.00	33.1
	SIMA$_{imd}$	0.862	0.941	0.900	0.671	**0.719**	**0.694**	32	28	1.00	307.6

Table 47: Comparative evaluation results for the public and proprietary logs - Inductive Miner.

Inductive Miner. Table 47 displays the results of our comparative evaluation for Inductive Miner. We held out from the table the five BPIC15 logs, because none of the three metaheuristics could discover a model within the five minutes timeout. This was due to scalability issues experienced by the Markovian accuracy, and already highlighted in Chapter 6 (for the case of IM).

In the remaining 15 logs, 13 times *all* the metaheuristics improved the Markovian F-score w.r.t. $\mathrm{IM_d}$, and only for the $\mathrm{BPIC17_f}$ log none of the metaheuristics could outperform $\mathrm{IM_d}$. The best performing metaheuristic was $\mathrm{SIMA_{imd}}$, achieving 8 times the highest Markovian F-score, followed by $\mathrm{TABU_{imd}}$ and $\mathrm{RLS_{imd}}$, who scored 7 and 6 times the highest Markovian F-score.

The results of the metaheuristics on the alignment-based F-score are similar to the case of Fodina, and they are broadly in-line with the results achieved on the Markovian F-score. Indeed, 80% of the times, *all* the metaheuristics were able to outperform $\mathrm{IM_d}$, failing only in two logs out of 15.

Regarding the complexity of the models discovered by the metaheuristics, we recorded little variation w.r.t. the complexity of the models discovered by $\mathrm{IM_d}$, meaning that the size and the CFC of the discovered models did not notably improve nor worsen in any case, expect for the PRT9 and the $\mathrm{BPIC14_f}$ logs, where both size and CFC were reduced of circa 30%.

In terms of execution times, the three metaheuristics perform similarly, with an average execution time close to 300 seconds, meaning that majority of the times the solution-space exploration was interrupted by the five minutes timeout.

7.3.3. Statistical Analysis

Finally, we used the Mann-Whitney U-test [6] to assess the statistical significance of our empirical evaluation. In particular, we analysed whether the improvements brought by our framework in terms of F-score were statistically significant. Table 48 reports the results of our statistical analysis, which can be summarised as follows. (i) Our optimization framework significantly improves the alignment-based and Markovian F-scores of $\mathrm{IM_d}$, regardless of the underlying metaheuristic. (ii) The alignment-based F-score improvements delivered by $\mathrm{RLS_{fo}}$ are significantly higher than those delivered by $\mathrm{HPO_{fo}}$. (iii) The Markovian F-score improvements of $\mathrm{SIMA_{sm}}$ are significantly higher than those of $\mathrm{HPO_{sm}}$. Event though we cannot claim that all the improvements of the alignment-based and Markovian F-scores of FO and SM achieved by the metaheuristics in our framework are statistical significant (i.e. with a confidence level greater than 95%), from the results in Table 48, we note that the confidence levels of the statistical results are always above 88.5%, except for the case of the alignment-based F-scores of SM.

Hypothesis: Distribution 1 significantly higher than Distribution 2.						
F-score Measure	Distribution 1	Distribution 2	$n_1 = n_2$	U_2	Z	Confidence
Alignment	RLS_{imd}	IM_{d}	15	58.5	2.24	97.5%
Alignment	TABU_{imd}	IM_{d}	15	61.5	2.12	96.6%
Alignment	SIMA_{imd}	IM_{d}	15	59.5	2.20	97.2%
Markovian	RLS_{imd}	IM_{d}	15	36.5	3.15	99.7%
Markovian	TABU_{imd}	IM_{d}	15	31	3.17	99.7%
Markovian	SIMA_{imd}	IM_{d}	15	34	3.26	99.8%
Alignment	RLS_{fo}	HPO_{fo}	15	64	2.01	95.6%
Alignment	ILS_{fo}	HPO_{fo}	15	74	1.60	89.0%
Alignment	TABU_{fo}	HPO_{fo}	15	74.5	1.58	88.6%
Alignment	SIMA_{fo}	HPO_{fo}	15	66	1.93	94.6%
Markovian	RLS_{fo}	HPO_{fo}	15	66.5	1.91	94.4%
Markovian	ILS_{fo}	HPO_{fo}	15	70.5	1.74	91.8%
Markovian	TABU_{fo}	HPO_{fo}	15	68.5	1.83	93.3%
Markovian	SIMA_{fo}	HPO_{fo}	15	67.5	1.87	93.9%
Alignment	RLS_{sm}	HPO_{sm}	16	106.5	0.81	58.2%
Alignment	ILS_{sm}	HPO_{sm}	16	108.5	0.74	54.1%
Alignment	TABU_{sm}	HPO_{sm}	16	111	0.64	47.8%
Alignment	SIMA_{sm}	HPO_{sm}	16	111.5	0.62	46.5%
Markovian	RLS_{sm}	HPO_{sm}	16	80	1.81	93.0%
Markovian	ILS_{sm}	HPO_{sm}	16	78	1.88	94.0%
Markovian	TABU_{sm}	HPO_{sm}	16	82	1.73	91.6%
Markovian	SIMA_{sm}	HPO_{sm}	16	75.5	1.98	95.2%

Table 48: Summary of the Mann-Whitney U-tests [6].

7.3.4. Discussion

The results of the experimental evaluation of our optimization framework applied to Split Miner, Fodina, and Inductive Miner, proved its effectiveness and efficiency. The metaheuristics integrated in our framework delivered different degrees of optimization depending on the underlying APDA, the complexity of the input event log, and the selected optimization metaheuristic, yet consistently outperforming the baselines in the vast majority of the cases (i.e. more than 80% of the times).

Overall, all the three APDAs successfully achieved better results in terms of Markovian F-score (our objective function) and alignment-based F-score when optimized through our framework, and only at the cost of a reasonably longer execution time (up to five minutes).

We note, however, that in a small number of cases the optimization framework could not bring any improvement to the accuracy achieved by the baseline APDAs due to: (i) a small solution-space (i.e. the baseline already discovers the best process model); or (ii) scalability issues (i.e. the Markovian accuracy cannot be computed within the five minutes timeout). While the former scenario is beyond our control and strictly relates to the complexity of the input event log, the latter reminds us of the limitations of the state-of-the-art accuracy measures (and especially precision) in the context of automated process discovery, and justifies our design choice of a modular optimization framework, that allows the implementation of (future) accuracy measures (as objective functions) able to overcome such scalability issues.

7.4. Summary

This chapter addressed our third and last research question, *how can the accuracy of an automated process discovery approach be efficiently optimized?* We proposed an optimization framework for DFG-based APDAs which is powered by single-solution-based optimization metaheuristics (S-metaheuristics), such as iterative local search, tabu search, and simulated annealing. The framework takes advantage of the DFG's simplicity to define efficient perturbation functions that allow the S-metaheuristics to explore the solution-space and discover process models with higher objective function scores. We designed our framework in a modular manner, allowing the integration of any DFG-based APDAs and objective function. In the context of this thesis, we instantiated the framework for Split Miner, Fodina, and Inductive Miner, since these are representative APDAs of the state of the art in automated process discovery, and we chose the Markovian F-score as objective function (introduced in Chapter 6).

The evaluation showed that when applying our framework to Split Miner, Fodina, and Inductive Miner, each of these three APDAs achieved higher accuracy for a clear majority of the events logs in the benchmark, particularly when using fine-grained measures of fitness and precision based on Markovian abstractions, but also when using measures based on alignments (especially in the case of Fodina and Inductive Miner). We note that these accuracy gains come at the expense of slightly higher execution times (up to five minutes) and sometimes greater model size and structural complexity (in the case of Split Miner).

Lastly, but most interesting, the experiments showed that the S-metaheuristics achieve higher accuracy than hyperparameter-optimized version of Split Miner and Fodina, while achieving lower execution times, highlighting the fact that our optimization framework enhances the underlying APDA, allowing it to explore areas of the solution-space that could not be reached by simply varying the input parameters.

8. CONCLUSION

8.1. Summary of Contributions

This thesis made four contributions to the field of automated discovery of business process models from event logs. The first contribution of this thesis is an extensive systematic literature review of the research studies addressing the problem of automated process discovery. We identified and analysed 34 studies proposing automated process discovery approaches (APDAs). Then, we designed a benchmark based on 24 real-life event logs and seven quality measures, and assessed a representative subset of the 34 APDAs. The outcome of this benchmark allowed us to identify strengths and weaknesses of the state-of-the-art APDAs, among the weaknesses, the following three were recurrent: (i) limited accuracy (i.e. low scores of fitness and/or precision); (ii) high computational and time complexity; (iii) syntactically incorrect outputs (i.e. unsound process models).

With the goal of overcoming these weaknesses, we designed a novel APDA, namely *Split Miner*, which represents our second (and core) contribution. Split Miner is the first APDA that guarantees to discover sound unstructured acyclic process models, and deadlock-free unstructured cyclic process models. Furthermore, Split Miner is designed to discover highly accurate process models, while maintaining a low computational and time complexity. We assessed Split Miner on our benchmark, and we showed that it outperforms the state-of-the-art APDAs, and successfully overcomes their three major weaknesses. Split Miner is integrated within the *process discoverer* plugin of Apromore, an open-source business process analytics platform providing the full suite of *process mining* functionalities, from automated process discovery to predictive process intelligence.[1] Such an integration, allowed our contribution to reach not only academics but also industry practitioners.[2].

Our extensive evaluations also highlighted that many state-of-the-art APDAs can achieve better results when their hyper-parameters are opportunely tuned. However, tuning the hyper-parameters of an APDA is a time consuming and computationally expensive exercise, due to two shortfalls in the state-of-ther-art: (i) the computational inefficiency of existing accuracy measures for automated process discovery, particularly precision measures; and (ii) the lack of efficient solution-space exploration methods for the problem of automated process discovery. The last two major contributions of this thesis focused on solving these two problems.

The third contribution of this thesis is a family of accuracy measures that address the first of the above shortcomings, our novel accuracy measures are based on the comparison of the Markovian abstractions of process models and event logs, hence the name Markovian accuracy. Such measures, despite approximate,

---

[1] `https://apromore.org`

[2] `https://apromore.org/testimonials/`

showed to be computationally faster than the state-of-the-art measures. Furthermore, we proved that they satisfy a set of desirable properties for accuracy measures that the state-of-the-art measures do not.

To address the second problem, we designed an optimization framework for APDAs powered by single-solution-based optimization metaheuristics, such as iterative local search, tabu search, and simulated annealing. Our optimization framework is modular and allows the integration of any APDA based on directly-follows graphs, as well as any objective function. In the context of this thesis, we set as objective function our Markovian accuracy F-score, and we instantiated our framework for three state-of-the-art APDAs (Split Miner, Fodina, and Inductive Miner). Finally, we empirically evaluated our framework, demonstrating that it allows an APDA to explore its solution-space in an efficient and effective manner beyond tuning the APDA hyper-parameters.

8.2. Future Work

Our most valuable contribution to the automated process discovery area, Split Miner, addressed the major problems that we identified surveying and benchmarking the research in the area. Among the open challenges identified, we note that none of the existing APDAs guarantees to discover sound unstructured cyclic process models. In fact, even though Split Miner can discover such process models, it does not guarantee this property. Our literature review and benchmark highlighted that the only APDAs that guarantees such a result imposes the restriction of discovering block-structured process models. This constraint can impair the accuracy of the discovered process models. Designing an APDA that guarantees sound unstructured cyclic process models is the natural extension of the work produced in this thesis.

Another open challenge, in automated process discovery, is the optimization of the state-of-the-art APDAs, in particular how to efficiently explore their solution-space in a fast and scalable manner. Our proposed optimization framework is a first step in this direction, but it currently relies on our Markovian accuracy measures, which despite faster than existing measures, they are approximate. Designing exact and scalable accuracy measures has proven very challenging. One of the latest studies proposes the use of the concept of entropy [166], showing that accuracy measures based on entropy are exact and satisfy even a stricter set of qualitative properties [4], yet they are not as efficient as our Markovian accuracy measures. Future work should consider the designing of exact and scalable accuracy measures, as well as the designing of alternative approaches to explore the solution-space of an APDA, ideally reducing the use of accuracy measures, for example by inferring the accuracy of a process model based on its location in the solution-space (e.g. analysing its neighbouring solutions).

Finally, an emerging research area closely related to process mining (and automated process discovery) is *robotic process automation* (RPA), which has recently attracted much attention from the research community. The ultimate goal of RPA is to automatically detect clerical routines performed by process participants (within a well-defined working environment), and train software-bots to replicate the detected automatable routines. Currently, the detection of automatable routines and the training of software-bots are time-consuming manual activities performed by process participants, while the research community of RPA aims to automate the whole pipeline as follows. First, the clerical routines performed by an user are (automatically) recorded into a so-called *user interaction log* (UI log). Next, the UI log is analysed to discover routines that can be automated [182]. Finally, the discovered automatable routines are compiled into software-bots. The discovery of automatable routines from UI logs is a problem very similar to automated process discovery of process models from event logs. Therefore, it would be interesting to explore the adaptation of APDAs to the discovery of automatable routines from UI logs, trying to overcome specific challenges of that context, among which: (i) avoiding to discover automatable routines that generalise the user interactions recorded in the input UI log; (ii) handling the noise in the UI log, as well as the variability of a given routine (i.e. the same routine can be performed in different ways); and (iii) dealing with complex data transformations (required to detect routines) in a scalable manner.

BIBLIOGRAPHY

[1] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of business process management*. Springer, 2013.

[2] B. van Dongen, J. Carmona, and T. Chatain, "A unified approach for measuring precision and generalization based on anti-alignments," in *BPM*, Springer, 2016.

[3] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs," *Information Systems*, vol. 37, no. 7, pp. 654–676, 2012.

[4] A. F. Syring, N. Tax, and W. M. van der Aalst, "Evaluating conformance measures in process mining using conformance propositions (extended version)," *arXiv preprint arXiv:1909.02393*, 2019.

[5] N. Tax, X. Lu, N. Sidorova, D. Fahland, and W. van der Aalst, "The imprecisions of precision measures in process mining," *Information Processing Letters*, vol. 135, 2018.

[6] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[7] W. van der Aalst, *Process Mining - Data Science in Action*. Springer, 2016.

[8] A. Weijters and J. Ribeiro, "Flexible heuristics miner (FHM)," in *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pp. 310–317, IEEE, 2011.

[9] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering Block-Structured Process Models from Incomplete Event Logs," in *Application and Theory of Petri Nets and Concurrency: 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings*, pp. 91–110, Springer International Publishing, 2014.

[10] J. C. Buijs, B. F. van Dongen, and W. M. van der Aalst, "Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity," *International Journal of Cooperative Information Systems*, vol. 23, no. 01, p. 1440001, 2014.

[11] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and G. Bruno, "Automated discovery of structured process models: Discover structured vs. discover and structure," in *Conceptual Modeling: 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings*, pp. 313–329, Springer, 2016.

[12] S. K. vanden Broucke and J. De Weerdt, "Fodina: a robust and flexible heuristic process discovery technique," *Decision Support Systems*, 2017.

[13] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *IEEE TKDE*, vol. 31, no. 4, 2019.

[14] J. Mendling, *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*. Springer, 2008.

[15] W. van der Aalst, K. van Hee, A. ter Hofstede, N. Sidorova, E. Verbeek, M. Voorhoeve, and M. Wynn, "Soundness of workflow nets: classification, decidability, and analysis," *Formal Asp. Comput.*, vol. 23, no. 3, 2011.

[16] R. H. Von Alan, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[17] A. Augusto, R. Conforti, M. Dumas, and M. La Rosa, "Split Miner: Discovering Accurate and Simple Business Process Models from Event Logs," in *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*, pp. 1–10, 2017.

[18] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and A. Polyvyanyy, "Split miner: automated discovery of accurate and simple business process models from event logs," *KAIS*, 2018.

[19] A. Augusto, A. Armas-Cervantes, R. Conforti, M. Dumas, M. La Rosa, and D. Reissner, "Abstract-and-compare: A family of scalable precision measures for automated process discovery," in *BPM*, Springer, 2018.

[20] A. Augusto, A. Armas Cervantes, R. Conforti, M. Dumas, M. La Rosa, and D. Reissner, "Measuring fitness and precision of automatically discovered process models: A principled and scalable approach," tech. rep., University of Melbourne, 2019.

[21] A. Augusto, M. Dumas, and M. La Rosa, "Metaheuristic optimization for automated business process discovery," in *BPM*, Springer, 2019.

[22] Object Management Group (OMG), *Business Process Model and Notation (BPMN) ver. 2.0*. Object Management Group (OMG), 1 2011.

[23] J. L. Peterson, "Petri nets," *ACM Computing Surveys (CSUR)*, vol. 9, no. 3, pp. 223–252, 1977.

[24] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[25] W. M. Van der Aalst, "The application of petri nets to workflow management," *Journal of circuits, systems, and computers*, vol. 8, no. 01, pp. 21–66, 1998.

[26] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas, "Structuring acyclic process models," *Inf. Syst.*, vol. 37, no. 6, pp. 518–538, 2012.

[27] W. van der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[28] B. Kitchenham, “Procedures for performing systematic reviews,” *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[29] A. Fink, *Conducting research literature reviews: from the internet to paper*. Sage Publications, 3rd edition ed., 2010.

[30] C. Okoli and K. Schabram, “A guide to conducting a systematic literature review of information systems research,” *Sprouts: Working Papers on Information Systems*, vol. 10, no. 26, pp. 1–49, 2010.

[31] J. Randolph, “A guide to writing the dissertation literature review,” *Practical Assessment, Research & Evaluation*, vol. 14, no. 13, pp. 1–13, 2009.

[32] R. Torraco, “Writing integrative literature reviews: guidelines and examples,” *Human Resource Development Review*, vol. 4, no. 3, pp. 356–367, 2005.

[33] W. Van der Aalst, T. Weijters, and L. Maruster, “Workflow mining: Discovering process models from event logs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[34] A. Alves de Medeiros, B. Van Dongen, W. Van Der Aalst, and A. Weijters, “Process mining: Extending the α-algorithm to mine short loops,” tech. rep., BETA Working Paper Series, 2004.

[35] L. Wen, W. M. van der Aalst, J. Wang, and J. Sun, “Mining process models with non-free-choice constructs,” *Data Mining and Knowledge Discovery*, vol. 15, no. 2, pp. 145–180, 2007.

[36] Q. Guo, L. Wen, J. Wang, Z. Yan, and S. Y. Philip, “Mining invisible tasks in non-free-choice constructs,” in *International Conference on Business Process Management*, pp. 109–125, Springer, 2015.

[37] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens, “Robust process discovery with artificial negative events,” *Journal of Machine Learning Research*, vol. 10, no. Jun, pp. 1305–1340, 2009.

[38] A. A. De Medeiros and A. Weijters, “Genetic process mining,” in *Applications and Theory of Petri Nets 2005, Volume 3536 of Lecture Notes in Computer Science*, Citeseer, 2005.

[39] A. K. A. de Medeiros, A. J. Weijters, and W. M. van der Aalst, “Genetic process mining: an experimental evaluation,” *Data Mining and Knowledge Discovery*, vol. 14, no. 2, pp. 245–304, 2007.

[40] A. J. Weijters and W. M. Van der Aalst, “Rediscovering workflow models from event-based data using little thumb,” *Integrated Computer-Aided Engineering*, vol. 10, no. 2, pp. 151–162, 2003.

[41] A. Weijters, W. M. van Der Aalst, and A. A. De Medeiros, “Process mining with the heuristics miner-algorithm,” *Technische Universiteit Eindhoven, Tech. Rep. WP*, vol. 166, pp. 1–34, 2006.

[42] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and G. Bruno, "Automated Discovery of Structured Process Models From Event Logs: The Discover-and-Structure Approach," *DKE*, 2017.

[43] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. van der Aalst, "Data-Driven Process Discovery-Revealing Conditional Infrequent Behavior from Event Logs," in *International Conference on Advanced Information Systems Engineering*, pp. 545–560, Springer, 2017.

[44] S. De Cnudde, J. Claes, and G. Poels, "Improving the quality of the heuristics miner in prom 6.2," *Expert Systems with Applications*, vol. 41, no. 17, pp. 7678–7690, 2014.

[45] J. M. E. van derWerf, B. F. van Dongen, C. A. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," *Fundamenta Informaticae*, vol. 94, no. 3-4, pp. 387–412, 2009.

[46] S. van Zelst, B. van Dongen, W. van der Aalst, and H. Verbeek, "Discovering workflow nets using integer linear programming," *Computing*, pp. 1–28, 2017.

[47] Z. Huang and A. Kumar, "A study of quality and accuracy trade-offs in process mining," *INFORMS Journal on Computing*, vol. 24, no. 2, pp. 311–327, 2012.

[48] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst, "Efficient Discovery of Understandable Declarative Process Models from Event Logs," in *Advanced Information Systems Engineering - 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings*, pp. 270–285, 2012.

[49] F. M. Maggi, A. J. Mooij, and W. M. van der Aalst, "User-guided discovery of declarative process models," in *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pp. 192–199, IEEE, 2011.

[50] F. M. Maggi, R. J. C. Bose, and W. M. van der Aalst, "A knowledge-based integrated approach for discovering and repairing declare maps," in *International Conference on Advanced Information Systems Engineering*, pp. 433–448, Springer, 2013.

[51] M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Discovering cross-organizational business rules from the cloud," in *2014 IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2014, Orlando, FL, USA, December 9-12, 2014*, pp. 389–396, 2014.

[52] F. M. Maggi, "Discovering metric temporal business constraints from event logs," in *Perspectives in Business Informatics Research - 13th International Conference, BIR 2014, Lund, Sweden, September 22-24, 2014. Proceedings*, pp. 261–275, 2014.

[53] T. Kala, F. M. Maggi, C. Di Ciccio, and C. Di Francescomarino, "Apriori and sequence analysis for discovering declarative process models," in

Enterprise Distributed Object Computing Conference (EDOC), 2016 IEEE 20th International, pp. 1–9, IEEE, 2016.

[54] F. M. Maggi, C. D. Ciccio, C. D. Francescomarino, and T. Kala, "Parallel algorithms for the automated discovery of declarative process models," *Information Systems*, 2017.

[55] C. Di Ciccio and M. Mecella, "A two-step fast algorithm for the automated discovery of declarative workflows," in *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*, pp. 135–142, IEEE, 2013.

[56] C. Di Ciccio and M. Mecella, "Mining constraints for artful processes," in *Business Information Systems - 15th International Conference, BIS 2012, Vilnius, Lithuania, May 21-23, 2012. Proceedings*, pp. 11–23, 2012.

[57] C. Di Ciccio and M. Mecella, "On the discovery of declarative control flows for artful processes," *ACM Trans. Management Inf. Syst.*, vol. 5, no. 4, pp. 24:1–24:37, 2015.

[58] C. Di Ciccio, F. M. Maggi, and J. Mendling, "Discovering target-branched declare constraints," in *International Conference on Business Process Management*, pp. 34–50, Springer, 2014.

[59] C. Di Ciccio, F. M. Maggi, and J. Mendling, "Efficient discovery of Target-Branched Declare Constraints," *Information Systems*, vol. 56, pp. 258–283, 2016.

[60] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, pp. 66–78, 2013.

[61] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering Block-Structured Process Models from Event Logs - A Constructive Approach," in *Application and Theory of Petri Nets and Concurrency: 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings*, pp. 311–329, Springer Berlin Heidelberg, 2013.

[62] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Exploring Processes and Deviations," in *Business Process Management Workshops: BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, pp. 304–316, Springer, 2014.

[63] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Using Life Cycle Information in Process Discovery," in *Business Process Management Workshops: BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 – September 3, 2015, Revised Papers*, pp. 204–217, Springer, 2015.

[64] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Scalable process discovery with guarantees," in *International Conference on Enterprise, Business-Process and Information Systems Modeling*, pp. 85–101, Springer, 2015.

[65] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Scalable process discovery and conformance checking," *Software & Systems Modeling*, pp. 1–33, 2016.

[66] M. Leemans and W. M. van der Aalst, "Modeling and discovering cancelation behavior," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pp. 93–113, Springer, 2017.

[67] F. M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali, "Discovering data-aware declarative process models from event logs," in *Business Process Management*, pp. 81–96, Springer, 2013.

[68] M. Abe and M. Kudo, "Business Monitoring Framework for Process Discovery with Real-Life Logs," in *International Conference on Business Process Management*, pp. 416–423, Springer, 2014.

[69] M. Kudo, A. Ishida, and N. Sato, "Business process discovery by using process skeletonization," in *Signal-Image Technology & Internet-Based Systems (SITIS), 2013 International Conference on*, pp. 976–982, IEEE, 2013.

[70] S. K. vanden Broucke, J. Vanthienen, and B. Baesens, "Declarative process discovery with evolutionary computing," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pp. 2412–2419, IEEE, 2014.

[71] W. M. van der Aalst, J. C. Buijs, and B. F. van Dongen, "Towards Improving the Representational Bias of Process Mining," in *Data-Driven Process Discovery and Analysis: First International Symposium (SIMPDA 2011)*, pp. 39–54, Springer, 2011.

[72] J. C. Buijs, B. F. van Dongen, and W. M. van der Aalst, "A genetic algorithm for discovering process trees," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pp. 1–8, IEEE, 2012.

[73] J. C. Buijs, B. F. Van Dongen, W. M. van Der Aalst, *et al.*, "On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery," in *OTM Conferences (1)*, vol. 7565, pp. 305–322, 2012.

[74] J. C. Buijs, B. F. van Dongen, and W. M. van der Aalst, "Discovering and navigating a collection of process models using multiple quality dimensions," in *Business Process Management Workshops: BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, pp. 3–14, Springer, 2013.

[75] M. L. van Eck, J. C. Buijs, and B. F. van Dongen, "Genetic Process Mining: Alignment-Based Process Model Mutation," in *Business Process Management Workshops: BPM 2014 International Workshops, Eindhoven, The*

Netherlands, September 7-8, 2014, Revised Papers, pp. 291–303, Springer, 2014.

[76] J. Carmona and J. Cortadella, "Process discovery algorithms using numerical abstract domains," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3064–3076, 2014.

[77] S. Ferilli, "Woman: logic-based workflow learning and management," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 6, pp. 744–756, 2014.

[78] S. Ferilli, B. De Carolis, and D. Redavid, "Logic-based incremental process mining in smart environments," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 392–401, Springer, 2013.

[79] B. De Carolis, S. Ferilli, and G. Mallardi, "Learning and Recognizing Routines and Activities in SOFiA," in *Ambient Intelligence: European Conference, AmI 2014, Eindhoven, The Netherlands, November 11-13, 2014. Revised Selected Papers*, pp. 191–204, Springer, 2014.

[80] S. Ferilli, B. De Carolis, and F. Esposito, "Learning Complex Activity Preconditions in Process Mining," in *New Frontiers in Mining Complex Patterns: Third International Workshop, NFMCP 2014, Held in Conjunction with ECML-PKDD 2014, Nancy, France, September 19, 2014, Revised Selected Papers*, pp. 164–178, Springer, 2014.

[81] S. Ferilli, D. Redavid, and F. Esposito, "Logic-Based Incremental Process Mining," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 218–221, Springer, 2015.

[82] S. Ferilli, "The WoMan Formalism for Expressing Process Models," in *Advances in Data Mining. Applications and Theoretical Aspects: 16th Industrial Conference, ICDM 2016, New York, NY, USA, July 13-17, 2016. Proceedings*, pp. 363–378, Springer, 2016.

[83] F. M. Maggi, T. Slaats, and H. A. Reijers, "The automated discovery of hybrid processes," in *International Conference on Business Process Management*, pp. 392–399, Springer, 2014.

[84] D. Redlich, T. Molka, W. Gilani, G. S. Blair, and A. Rashid, "Scalable Dynamic Business Process Discovery with the Constructs Competition Miner," in *Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014)*, pp. 91–107, 2014.

[85] D. Redlich, T. Molka, W. Gilani, G. Blair, and A. Rashid, "Constructs competition miner: Process control-flow discovery of bp-domain constructs," in *International Conference on Business Process Management*, pp. 134–150, Springer, 2014.

[86] D. Redlich, W. Gilani, T. Molka, M. Drobek, A. Rashid, and G. Blair, "Introducing a framework for scalable dynamic process discovery," in *En-*

terprise Engineering Working Conference, pp. 151–166, Springer, 2014.

[87] D. Redlich, T. Molka, W. Gilani, G. Blair, and A. Rashid, "Dynamic constructs competition miner-occurrence-vs. time-based ageing," in *International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA 2014)*, pp. 79–106, Springer, 2014.

[88] O. Vasilecas, T. Savickas, and E. Lebedys, "Directed acyclic graph extraction from event logs," in *International Conference on Information and Software Technologies*, pp. 172–181, Springer, 2014.

[89] J. De Smedt, J. De Weerdt, and J. Vanthienen, "Fusion miner: process discovery for mixed-paradigm models," *Decision Support Systems*, vol. 77, pp. 123–136, 2015.

[90] G. Greco, A. Guzzo, F. Lupia, and L. Pontieri, "Process discovery under precedence constraints," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 9, no. 4, p. 32, 2015.

[91] G. Greco, A. Guzzo, and L. Pontieri, "Process discovery via precedence constraints," in *Proceedings of the 20th European Conference on Artificial Intelligence*, pp. 366–371, IOS Press, 2012.

[92] V. Liesaputra, S. Yongchareon, and S. Chaisiri, "Efficient process model discovery using maximal pattern mining," in *International Conference on Business Process Management*, pp. 441–456, Springer, 2015.

[93] T. Molka, D. Redlich, M. Drobek, X.-J. Zeng, and W. Gilani, "Diversity guided evolutionary mining of hierarchical process models," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 1247–1254, ACM, 2015.

[94] B. Vázquez-Barreiros, M. Mucientes, and M. Lama, "Prodigen: Mining complete, precise and minimal structure process models with a genetic algorithm," *Information Sciences*, vol. 294, pp. 315–333, 2015.

[95] B. Vázquez-Barreiros, M. Mucientes, and M. Lama, "A genetic algorithm for process discovery guided by completeness, precision and simplicity," in *International Conference on Business Process Management*, pp. 118–133, Springer, 2014.

[96] M. L. Bernardi, M. Cimitile, C. Di Francescomarino, and F. M. Maggi, "Do activity lifecycles affect the validity of a business rule in a business process?," *Information Systems*, 2016.

[97] M. L. Bernardi, M. Cimitile, C. D. Francescomarino, and F. M. Maggi, "Using discriminative rule mining to discover declarative process models with non-atomic activities," in *Rules on the Web. From Theory to Applications - 8th International Symposium, RuleML 2014, Co-located with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18-20, 2014. Proceedings*, pp. 281–295, 2014.

[98] D. Breuker, M. Matzner, P. Delfmann, and J. Becker, “Comprehensible predictive models for business processes,” *MIS Quarterly*, vol. 40, no. 4, pp. 1009–1034, 2016.

[99] D. Breuker, P. Delfmann, M. Matzner, and J. Becker, “Designing and evaluating an interpretable predictive modeling technique for business processes,” in *Business Process Management Workshops: BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, pp. 541–553, Springer, 2014.

[100] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, “Bpmn miner: Automated discovery of bpmn process models with hierarchical structure,” *Information Systems*, vol. 56, pp. 284–303, 2016.

[101] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, “Beyond tasks and gateways: Discovering bpmn models with subprocesses, boundary events and activity markers,” in *International Conference on Business Process Management*, pp. 101–117, Springer, 2014.

[102] M. L. van Eck, N. Sidorova, and W. M. van der Aalst, “Discovering and exploring state-based models for multi-perspective processes,” in *International Conference on Business Process Management*, pp. 142–157, Springer, 2016.

[103] M. L. van Eck, N. Sidorova, and W. M. P. van der Aalst, “Guided Interaction Exploration in Artifact-centric Process Models,” in *19th IEEE Conference on Business Informatics, CBI 2017, Thessaloniki, Greece, July 24-27, 2017, Volume 1: Conference Papers*, pp. 109–118, 2017.

[104] C. Li, J. Ge, L. Huang, H. Hu, B. Wu, H. Yang, H. Hu, and B. Luo, “Process mining with token carried data,” *Information Sciences*, vol. 328, pp. 558–576, 2016.

[105] A. Mokhov, J. Carmona, and J. Beaumont, “Mining Conditional Partial Order Graphs from Event Logs,” in *Transactions on Petri Nets and Other Models of Concurrency XI*, pp. 114–136, Springer, 2016.

[106] S. Schönig, A. Rogge-Solti, C. Cabanillas, S. Jablonski, and J. Mendling, “Efficient and customisable declarative process mining with sql,” in *International Conference on Advanced Information Systems Engineering*, pp. 290–305, Springer, 2016.

[107] S. Schönig, C. Di Ciccio, F. M. Maggi, and J. Mendling, “Discovery of multi-perspective declarative process models,” in *International Conference on Service-Oriented Computing*, pp. 87–103, Springer, 2016.

[108] W. Song, H.-A. Jacobsen, C. Ye, and X. Ma, “Process discovery from dependence-complete event logs,” *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 714–727, 2016.

[109] T. Tapia-Flores, E. Rodríguez-Pérez, and E. López-Mellado, “Discovering Process Models from Incomplete Event Logs using Conjoint Occurrence

Classes," in *ATAED@ Petri Nets/ACSD*, pp. 31–46, 2016.

[110] B. N. Yahya, M. Song, H. Bae, S.-o. Sul, and J.-Z. Wu, "Domain-driven actionable process model discovery," *Computers & Industrial Engineering*, 2016.

[111] B. N. Yahya, H. Bae, S.-o. Sul, and J.-Z. Wu, "Process discovery by synthesizing activity proximity and user's domain knowledge," in *Asia-Pacific Conference on Business Process Management*, pp. 92–105, Springer, 2013.

[112] J. De Weerdt, S. K. vanden Broucke, and F. Caron, "Bidimensional Process Discovery for Mining BPMN Models," in *Business Process Management Workshops: BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, pp. 529–540, Springer, 2014.

[113] H. Nguyen, M. Dumas, A. H. ter Hofstede, M. La Rosa, and F. M. Maggi, "Mining business process stages from event logs," in *International Conference on Advanced Information Systems Engineering*, pp. 577–594, Springer, 2017.

[114] H. Verbeek, W. van der Aalst, and J. Munoz-Gama, "Divide and Conquer: A Tool Framework for Supporting Decomposed Discovery in Process Mining," *The Computer Journal*, pp. 1–26, 2017.

[115] H. Verbeek and W. van der Aalst, "An experimental evaluation of passage-based process discovery," in *Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2012)*, vol. 132, pp. 205–210, 2012.

[116] W. M. Van der Aalst, "Decomposing Petri nets for process mining: A generic approach," *Distributed and Parallel Databases*, vol. 31, no. 4, pp. 471–507, 2013.

[117] B. Hompes, H. Verbeek, and W. M. van der Aalst, "Finding suitable activity clusters for decomposed process discovery," in *International Symposium on Data-Driven Process Discovery and Analysis*, pp. 32–57, Springer, 2014.

[118] H. Verbeek and W. M. van der Aalst, "Decomposed Process Mining: The ILP Case," in *Business Process Management Workshops: BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, pp. 264–276, Springer, 2014.

[119] W. M. van der Aalst and H. Verbeek, "Process discovery and conformance checking using passages," *Fundamenta Informaticae*, vol. 131, no. 1, pp. 103–138, 2014.

[120] S. J. van Zelst, B. F. van Dongen, and W. M. van der Aalst, "Avoiding over-fitting in ILP-based process discovery," in *International Conference on Business Process Management*, pp. 163–171, Springer, 2015.

[121] S. J. van Zelst, B. F. van Dongen, and W. M. P. van der Aalst, "ILP-Based Process Discovery Using Hybrid Regions," in *International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015*,

vol. 1371 of *CEUR Workshop Proceedings*, pp. 47–61, CEUR-WS.org, 2015.

[122] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.

[123] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, "DECLARE: full support for loosely-structured processes," in *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, Maryland, USA*, pp. 287–300, 2007.

[124] M. Westergaard and F. M. Maggi, "Declare: A tool suite for declarative workflow modeling and enactment," in *Proceedings of the Demo Track of the Nineth Conference on Business Process Management 2011, Clermont-Ferrand, France, August 31st, 2011*, 2011.

[125] T. Slaats, D. M. M. Schunselaar, F. M. Maggi, and H. A. Reijers, *The Semantics of Hybrid Process Models*, pp. 531–551. 2016.

[126] M. Westergaard and T. Slaats, "Mixing paradigms for more comprehensible models," in *BPM*, pp. 283–290, 2013.

[127] C. Favre, D. Fahland, and H. Völzer, "The relationship between workflow graphs and free-choice workflow nets," *Inf. Syst.*, vol. 47, pp. 197–219, 2015.

[128] A. Adriansyah, B. van Dongen, and W. van der Aalst, "Conformance checking using cost-based fitness analysis," in *EDOC*, IEEE, 2011.

[129] A. Adriansyah, J. Muñoz-Gama, J. Carmona, B. van Dongen, and W. van der Aalst, "Alignment based precision checking," in *BPM Workshops*, LNBIP 132, Springer, 2012.

[130] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, pp. 1137–1145, Morgan Kaufmann, 1995.

[131] A. Rozinat, A. Alves de Medeiros, C. G´unther, A. Weijters, and W. van der Aalst, "Towards an evaluation framework for process mining algorithms," bpm center report bpm-07-06, 2007.

[132] A. Bolt, M. de Leoni, and W. M. P. van der Aalst, "Scientific workflows for process mining: building blocks, scenarios, and implementation," *Software Tools and Technology Transfer*, vol. 18, no. 6, pp. 607–628, 2016.

[133] B. F. van Dongen, J. Carmona, T. Chatain, and F. Taymouri, "Aligning modeled and observed behavior: A compromise between computation complexity and quality," in *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017*, pp. 94–109, Springer, 2017.

[134] J. Mendling, H. Reijers, and W. van der Aalst, "Seven process modeling guidelines (7PMG)," *Information and Software Technology*, vol. 52, no. 2, pp. 127–136, 2010.

[135] J. S. Cardoso, “Business process control-flow complexity: Metric, evaluation, and validation,” *Int. J. Web Service Res.*, vol. 5, no. 2, pp. 49–76, 2008.

[136] W. M. P. van der Aalst, *Verification of workflow nets*, pp. 407–426. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997.

[137] R. Conforti, M. L. Rosa, and A. ter Hofstede, “Filtering out infrequent behavior from business process event logs,” *IEEE TKDE*, vol. 29, no. 2, 2017.

[138] van der Aalst, W. M. P., van Dongen, B. F., J. Herbst, L. Maruster, G. Schimm, and Weijters, A. J. M. M., “Workflow mining: a survey of issues and approaches,” *Data Knowl. Eng.*, vol. 47, no. 2, pp. 237–267, 2003.

[139] J. Claes and G. Poels, “Process Mining and the ProM Framework: An Exploratory Survey,” in *Business Process Management Workshops*, pp. 187–198, Springer, 2012.

[140] S. K. L. M. vanden Broucke, J. D. Weerdt, J. Vanthienen, and B. Baesens, “A comprehensive benchmarking framework (CoBeFra) for conformance analysis between procedural process models and event logs in ProM,” in *IEEE Symposium on Computational Intelligence and Data Mining, CIDM*, pp. 254–261, IEEE, 2013.

[141] W. van der Aalst, T. Weijters, and L. Maruster, “Workflow mining: Discovering process models from event logs,” *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, 2004.

[142] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 12 1959.

[143] A. Polyvyanyy, J. Vanhatalo, and H. Völzer, “Simplified computation and generalization of the refined process structure tree,” in *WS-FM*, pp. 25–41, 2010.

[144] H. Völzer, “A new semantics for the inclusive converging gateway in safe processes,” in *International Conference on Business Process Management*, pp. 294–309, Springer, 2010.

[145] G. Janssenswillen, N. Donders, T. Jouck, and B. Depaire, “A comparative study of existing quality measures for process discovery,” *Information Systems*, vol. 71, pp. 1–15, 2017.

[146] A. Polyvyanyy, A. Solti, M. Weidlich, C. D. Ciccio, and J. Mendling, “Monotone precision and recall measures for comparing executions and specifications of dynamic systems,” *CoRR*, vol. abs/1812.07334, 2018.

[147] W. M. P. van der Aalst, “Relating process models and event logs - 21 conformance propositions,” in *International Workshop on Algorithms & Theories for the Analysis of Event Data*, pp. 56–74, Springer, 2018.

[148] A. Rozinat and W. M. Van der Aalst, "Conformance testing: Measuring the fit and appropriateness of event logs and process models," in *International Conference on Business Process Management*, pp. 163–176, Springer, 2005.

[149] A. K. A. de Medeiros, *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, 2006.

[150] S. K. vanden Broucke, J. Munoz-Gama, J. Carmona, B. Baesens, and J. Vanthienen, "Event-based real-time decomposed conformance analysis," in *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pp. 345–363, Springer, 2014.

[151] A. Adriansyah, B. F. van Dongen, and W. M. van der Aalst, "Conformance checking using cost-based fitness analysis," in *IEEE International on Enterprise Distributed Object Computing Conference (EDOC)*, pp. 55–64, IEEE, 2011.

[152] S. Leemans, D. Fahland, and W. van der Aalst, "Scalable process discovery and conformance checking," *Software & Systems Modeling*, 2016.

[153] T. Ceccherini-Silberstein, A. Machi, and F. Scarabotti, "On the entropy of regular languages," *Theoretical computer science*, vol. 307, no. 1, pp. 93–102, 2003.

[154] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, "Discovering expressive process models by clustering log traces," *IEEE TKDE*, vol. 18, no. 8, 2006.

[155] A. Rozinat and W. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *ISJ*, vol. 33, no. 1, 2008.

[156] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A robust f-measure for evaluating discovered process models," in *IEEE Symposium on CIDM*, IEEE, 2011.

[157] J. Munoz-Gama and J. Carmona, "A fresh look at precision in process conformance," in *BPM*, Springer, 2010.

[158] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, and W. van der Aalst, "Measuring precision of modeled behavior," *ISeB*, vol. 13, no. 1, 2015.

[159] S. Leemans, D. Fahland, and W. van der Aalst, "Discovering block-structured process models from event logs - a constructive approach," in *Petri Nets*, Springer, 2013.

[160] S. M. Ross, *Introduction to probability models*. Academic press, 2014.

[161] H. Kuhn, "The hungarian method for the assignment problem," *NRL*, vol. 2, no. 1-2, 1955.

[162] H. W. Kuhn, "Variants of the hungarian method for assignment problems," *Naval Research Logistics Quarterly*, vol. 3, no. 4, pp. 253–258, 1956.

[163] J. Munkres, “Algorithms for the assignment and transportation problems,” *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[164] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” in *Soviet physics doklady*, vol. 10, pp. 707–710, 1966.

[165] A. Backurs and P. Indyk, “Edit distance cannot be computed in strongly subquadratic time (unless SETH is false),” in *47th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 51–58, ACM, 2015.

[166] A. Polyvyanyy and A. Kalenkova, “Monotone conformance checking for partially matching designed and observed processes,” in *International Conference on Process Mining*, pp. 81–88, 06 2019.

[167] B. F. van Dongen, “Efficiently computing alignments - using the extended marking equation,” in *International Conference on Business Process Management (BPM)*, pp. 197–214, Springer, 2018.

[168] J. Buijs, B. van Dongen, and W. van der Aalst, “On the role of fitness, precision, generalization and simplicity in process discovery,” in *CoopIS*, Springer, 2012.

[169] W. Song, S. Liu, and Q. Liu, “Business process mining based on simulated annealing,” in *ICYCS*, IEEE, 2008.

[170] D. Gao and Q. Liu, “An improved simulated annealing algorithm for process mining,” in *CSCWD*, IEEE, 2009.

[171] I. Boussaïd, J. Lepagnot, and P. Siarry, “A survey on optimization metaheuristics,” *Information Sciences*, vol. 237, 2013.

[172] T. Stützle, *Local search algorithms for combinatorial problems*. PhD thesis, Darmstadt University of Technology, 1998.

[173] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Computers & operations research*, vol. 13, no. 5, 1986.

[174] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *science*, vol. 220, no. 4598, 1983.

[175] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[176] T. Bäck and H.-P. Schwefel, “An overview of evolutionary algorithms for parameter optimization,” *Evolutionary computation*, vol. 1, no. 1, pp. 1–23, 1993.

[177] J. Kennedy, “Particle swarm optimization,” in *Encyclopedia of machine learning*, pp. 760–766, Springer, 2011.

[178] E. Atashpaz-Gargari and C. Lucas, “Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition,” in *Evolutionary computation, 2007. CEC 2007. IEEE Congress on*, pp. 4661–4667, IEEE, 2007.

[179] S. Alizadeh and A. Norani, “Icma: a new efficient algorithm for process model discovery,” *Applied Intelligence*, vol. 48, no. 11, 2018.

[180] V. R. Chifu, C. B. Pop, I. Salomie, I. Balla, and R. Paven, “Hybrid particle swarm optimization method for process mining,” in *ICCP*, IEEE, 2012.

[181] A. Adriansyah, B. van Dongen, and W. van der Aalst, “Conformance checking using cost-based fitness analysis,” in *EDOC*, IEEE, 2011.

[182] A. Bosco, A. Augusto, M. Dumas, M. La Rosa, and G. Fortino, “Discovering automatable routines from user interaction logs,” in *International Conference on Business Process Management*, pp. 144–162, Springer, 2019.

ACKNOWLEDGEMENT

The outcomes of our endeavours are direct consequences of ***all*** the actions and decisions we have taken along the way, whether little or big. For this reason, I am grateful to all the people I met, talked, and worked with; among them especially to Marcello and Marlon, my mentors; to Mia, my partner; to Maria Concetta and Antonino, my parents. Also, I am grateful to all the places, cities, and countries that welcomed, hosted, and inspired me; among them especially Estonia and Australia, the former because allowed me to appreciate the beauty of a small community, the latter because allowed me to find a new home.

SUMMARY IN ESTONIAN

Täpne ja tõhus protsessimudelite automaatne koostamine sündmuslogidest

Firmad üle kogu maailma tarnivad iga päev tooteid ja teenuseid keerukate äriprotsesside mudelite täitmise abil. Vastavate äriprotsesside kvaliteet ja tõhusus mõjutavad otseselt kasutajakogemust. Seetõttu on firmadele väga oluline hoolikalt hallata oma äriprotsesse.

Äriprotsesside haldamiseks on vajalikud nende protsesside üksikasjalikud mudelid. Traditsiooniliselt kavandatakse äriprotsesside mudelid käsitsi. Protsessimudelite käsitsi kavandamine võtab aga palju aega. See nõuab, et analüütikud koguksid protsessi täitmise kohta üksikasjalikku teavet. Peale selle on käsitsi kavandatud protsessimudelid sageli mittetäielikud, sest käsitsi andmekogumise abil on raske koostada protsessi kõikvõimalikke täitmisteid.

Äriprotsesside mudelite automaatse koostamise meetodid lubavad analüütikutel kasutada andmeid äriprotsesside täitmise kohta (mis on salvestatud nn sündmuslogidesse) protsessimudelite automaatseks genereerimiseks. Käesolevas väitekirjas teeme süstemaatilise kirjanduse ülevaate ja viime läbi kõige uuemate protsessimudelite automaatsete koostamise meetodite võrdleva analüüsi. See analüüs näitab, et praegustel antud valdkonna meetoditel on kolm ühist piirangut: (i) piiratud täpsus; (ii) arvutuslik ebatõhusus tegelikus elus kasutamiseks; (iii) süntaktiliselt vigaste protsessimudelite genereerimine.

Nimetatud piirangutest ülesaamiseks paneme ette uue meetodi protsessimudelite automaatseks koostamiseks, mille nimetuseks on Split Miner. Näitame empiiriliselt, et uus meetod ületab kõik muud kaasaegseimad meetodid nii täpsuse kui ka tõhususe poolest. Samuti näitame, et on võimalik isegi veelgi enam tõsta nii meetodi Split Miner kui ka teiste meetodite abil genereeritud protsessimudelite täpsust, kasutades selleks optimeerimisalgoritme. Siiski tuleb niisugune täpsuse paranemine palju aeglasema täitmisaja arvelt.

Üks põhjusi, miks olemasolevate protsessimudelite automaatse koostamise meetodite täpsuse optimeerimine ei ole tõhus seisneb selles, et praegused automaatselt koostatud protsessimudelite täpsuse hindamise mõõdikud on arvutuslikult liiga aeganõudvad. Sellele lisaks annavad olemasolevad täpsusmõõdikud sageli intuitsioonile mittevastavaid tulemusi. Kirjeldatud probleemi lahendamiseks paneme ette uue kogumi täpsusmõõdikuid, mis põhinevad Markovi mudelitel. Näitame, et neid täpsusmõõdikud saab tõhusalt välja arvutada ja need on olemasolevatest täpsusmõõdikutest intuitiivsemad ja neil on hulk soovitud omadusi.

Nimetatud täpsusmõõdikute alusel paneme ette optimeerimisraamistiku, mis põhineb optimeerimise metaheuristikal nagu näiteks “mäkkeronimise” või simuleeritud lõõmutamismeetodi algoritmid. Näitame, et see optimeerimisraamistik võimaldab oluliselt parandada olemasolevate automatiseeritud protsessimudelite automaatse koostamise meetodite täpsust ja selle arvutuslik tõhusus on piisav tegeliku elu stsenaariumites kasutamiseks.

Väitekiri koosneb kaheksast peatükist. Kahes esimeses peatükis antakse ülevaade probleemist, mida väitekirjas käsitletakse. Kolmandas peatükis antakse süstemaatiline ülevaade olemasolevaid protsessimudelite automaatse koostamise meetodeid käsitlevast kirjandusest. Neljandas peatükis esitatakse nende tehnikate katsepõhine hindamine. Kolmes järgnevad peatükis tutvustatakse Split Miner algoritmi, täpsusmõõdikuid ja optimeerimisalgoritme. Viimases peatükis tehakse kokkuvõte väitekirjas antud panustest ja kirjeldatakse üldjoontes tulevase töö suundi.

CURRICULUM VITAE

Personal data

Name: Adriano Augusto
Citizenship: Italian

Education

2016–2020	joint doctor of philosophy programme in computer science – University of Tartu and University of Melbourne
2013–2016	master's degree in computer engineering – Politecnico di Torino
2009–2013	bachelor's degree in computer engineering – Politecnico di Torino

Employment

2019–	associate lecturer – University of Melbourne
2015–2016	research assistant – Queensland University of Technology

Scientific work

Main fields of interest:

- process mining
- automated process discovery
- robotic process automation

ELULOOKIRJELDUS

Isikuandmed

Nimi: Adriano Augusto
Kodakondsus: Itaalia

Haridus

2016–2020	Tartu Ülikooli ja Melbourne'i Ülikooli ühine doktoriõpe informaatika erialal
2013–2016	Torino Polütehnikum, arvutitehnika magistriõpe
2009–2013	Torino Polütehnikum, arvutitehnika bakalaureuseõpe

Teenistuskäik

2019–	Melbourne'i Ülikooli, abilektor
2015–2016	Queenslandi Tehnikaülikool, nooremteadur

Teadustegevus

Peamised uurimisvaldkonnad:

- protsessikaeve
- protsessimudelite automaatne koostamine
- robotprotsesside automatiseerimine

LIST OF ORIGINAL PUBLICATIONS

This thesis is the final outcome of several research studies completed over the past three years. This section lists the peer-reviewed journal articles and conference papers published during my PhD, as well as technical reports yet to be published. The following research studies are integrated as chapters into this thesis, and acknowledged as footnotes at the beginning of the corresponding chapters.

Publications

- Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Maggi, F. M., Marrella, A., Mecella, M., & Soo, A. (2018). Automated discovery of process models from event logs: Review and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 31(4), 686-705.
- Augusto, A., Conforti, R., Dumas, M., & La Rosa, M. (2017, November). Split miner: Discovering accurate and simple business process models from event logs. In *IEEE International Conference on Data Mining (ICDM) 2017*, (pp. 1-10). IEEE.
- Augusto, A., Conforti, R., Dumas, M., La Rosa, M., & Polyvyanyy, A. (2019). Split miner: automated discovery of accurate and simple business process models from event logs. *Knowledge and Information Systems*, 59(2), 251-284.
- Augusto, A., Armas-Cervantes, A., Conforti, R., Dumas, M., La Rosa, M., & Reissner, D. (2018, September). Abstract-and-Compare: A Family of Scalable Precision Measures for Automated Process Discovery. In *International Conference on Business Process Management (BPM) 2018* (pp. 158-175). Springer.
- Augusto, A., Dumas, M., & La Rosa, M. (2019, September). Metaheuristic Optimization for Automated Business Process Discovery. In *International Conference on Business Process Management (BPM) 2019*. Springer.

Technical Reports

- Augusto, A., Armas-Cervantes, A., Conforti, R., Dumas, M., & La Rosa, M. (2018). Measuring Fitness and Precision of Automatically Discovered Process Models: a Principled and Scalable Approach. Submitted to *IEEE Transactions on Knowledge and Data Engineering*.
- Augusto, A., Dumas, M., La Rosa, M., Leemans, S. J. J., & vanden Broucke, S. K. L. M. (2019). Optimized Discovery of Process Models From Event Logs: Framework and Evaluation. Submitted to *Data & Knowledge Engineering*.

DISSERTATIONES INFORMATICAE PREVIOUSLY PUBLISHED IN DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 lk.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.

74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.
78. **Reina Uba.** Merging business process models. Tartu 2011, 166 p.
79. **Uuno Puus.** Structural performance as a success factor in software development projects – Estonian experience. Tartu 2012, 106 p.
81. **Georg Singer.** Web search engines and complex information needs. Tartu 2012, 218 p.
83. **Dan Bogdanov.** Sharemind: programmable secure computations with practical applications. Tartu 2013, 191 p.
84. **Jevgeni Kabanov.** Towards a more productive Java EE ecosystem. Tartu 2013, 151 p.
87. **Margus Freudenthal.** Simpl: A toolkit for Domain-Specific Language development in enterprise information systems. Tartu, 2013, 151 p.
90. **Raivo Kolde.** Methods for re-using public gene expression data. Tartu, 2014, 121 p.
91. **Vladimir Šor.** Statistical Approach for Memory Leak Detection in Java Applications. Tartu, 2014, 155 p.
92. **Naved Ahmed.** Deriving Security Requirements from Business Process Models. Tartu, 2014, 171 p.
94. **Liina Kamm.** Privacy-preserving statistical analysis using secure multi-party computation. Tartu, 2015, 201 p.
100. **Abel Armas Cervantes.** Diagnosing Behavioral Differences between Business Process Models. Tartu, 2015, 193 p.
101. **Fredrik Milani.** On Sub-Processes, Process Variation and their Interplay: An Integrated Divide-and-Conquer Method for Modeling Business Processes with Variation. Tartu, 2015, 164 p.
102. **Huber Raul Flores Macario.** Service-Oriented and Evidence-aware Mobile Cloud Computing. Tartu, 2015, 163 p.
103. **Tauno Metsalu.** Statistical analysis of multivariate data in bioinformatics. Tartu, 2016, 197 p.
104. **Riivo Talviste.** Applying Secure Multi-party Computation in Practice. Tartu, 2016, 144 p.
108. **Siim Orasmaa.** Explorations of the Problem of Broad-coverage and General Domain Event Analysis: The Estonian Experience. Tartu, 2016, 186 p.
109. **Prastudy Mungkas Fauzi.** Efficient Non-interactive Zero-knowledge Protocols in the CRS Model. Tartu, 2017, 193 p.
110. **Pelle Jakovits.** Adapting Scientific Computing Algorithms to Distributed Computing Frameworks. Tartu, 2017, 168 p.
111. **Anna Leontjeva.** Using Generative Models to Combine Static and Sequential Features for Classification. Tartu, 2017, 167 p.
112. **Mozhgan Pourmoradnasseri.** Some Problems Related to Extensions of Polytopes. Tartu, 2017, 168 p.

113. **Jaak Randmets.** Programming Languages for Secure Multi-party Computation Application Development. Tartu, 2017, 172 p.
114. **Alisa Pankova.** Efficient Multiparty Computation Secure against Covert and Active Adversaries. Tartu, 2017, 316 p.
116. **Toomas Saarsen.** On the Structure and Use of Process Models and Their Interplay. Tartu, 2017, 123 p.
121. **Kristjan Korjus.** Analyzing EEG Data and Improving Data Partitioning for Machine Learning Algorithms. Tartu, 2017, 106 p.
122. **Eno Tõnisson.** Differences between Expected Answers and the Answers Offered by Computer Algebra Systems to School Mathematics Equations. Tartu, 2017, 195 p.

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

1. **Abdullah Makkeh**. Applications of Optimization in Some Complex Systems. Tartu 2018, 179 p.
2. **Riivo Kikas**. Analysis of Issue and Dependency Management in Open-Source Software Projects. Tartu 2018, 115 p.
3. **Ehsan Ebrahimi**. Post-Quantum Security in the Presence of Superposition Queries. Tartu 2018, 200 p.
4. **Ilya Verenich**. Explainable Predictive Monitoring of Temporal Measures of Business Processes. Tartu 2019, 151 p.
5. **Yauhen Yakimenka**. Failure Structures of Message-Passing Algorithms in Erasure Decoding and Compressed Sensing. Tartu 2019, 134 p.
6. **Irene Teinemaa**. Predictive and Prescriptive Monitoring of Business Process Outcomes. Tartu 2019, 196 p.
7. **Mohan Liyanage.** A Framework for Mobile Web of Things. Tartu 2019, 131 p.
8. **Toomas Krips.** Improving performance of secure real-number operations. Tartu 2019, 146 p.
9. **Vijayachitra Modhukur.** Profiling of DNA methylation patterns as biomarkers of human disease. Tartu 2019, 134 p.
10. **Elena Sügis.** Integration Methods for Heterogeneous Biological Data. Tartu 2019, 250 p.
11. **Tõnis Tasa.** Bioinformatics Approaches in Personalised Pharmacotherapy. Tartu 2019, 150 p.
12. **Sulev Reisberg.** Developing Computational Solutions for Personalized Medicine. Tartu 2019, 126 p.
13. **Huishi Yin.** Using a Kano-like Model to Facilitate Open Innovation in Requirements Engineering. Tartu 2019, 129 p.
14. **Faiz Ali Shah.** Extracting Information from App Reviews to Facilitate Software Development Activities. Tartu 2020, 149 p.