Master Thesis

Developing Heuristic Fallthroughs for the Inductive Miner

Calvin Schröder
Matriculation Number: 440326

Examiners:
Prof. Sander J. J. Leemans
Prof. Wil M.P. van der Aalst

Supervisor:
Jan Niklas van Detten

Aachen, February 17, 2025

The present work was submitted to the i9 Chair of RWTH Aachen University

Eidesstattliche Versicherung
Declaration of Academic Integrity

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)
Student ID Number (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare under penalty of perjury that I have completed the present paper/bachelor's thesis/master's thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt; dies umfasst insbesondere auch Software und Dienste zur Sprach-, Text- und Medienproduktion. Ich erkläre, dass für den Fall, dass die Arbeit in unterschiedlichen Formen eingereicht wird (z.B. elektronisch, gedruckt, geplottet, auf einem Datenträger) alle eingereichten Versionen vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without unauthorized assistance from third parties (in particular academic ghostwriting**.** I have not used any other sources or aids than those indicated; this includes in particular software and services for language, text, and media production. In the event that the work is submitted in different formats (e.g. electronically, printed, plotted, on a data carrier), I declare that all the submitted versions are fully identical. I have not previously submitted this work, either in the same or a similar form to an examination body.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen/Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
§ 156 StGB (German Criminal Code): False Unsworn Declarations
Whosoever before a public authority competent to administer unsworn declarations (including Declarations of Academic Integrity) falsely submits such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment for a term not exceeding three years or to a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
§ 161 StGB (German Criminal Code): False Unsworn Declarations Due to Negligence
(1) If an individual commits one of the offenses listed in §§ 154 to 156 due to negligence, they are liable to imprisonment for a term not exceeding one year or to a fine.
(2) The offender shall be exempt from liability if they correct their false testimony in time. The provisions of § 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature

Abstract

Business process optimization typically involves discovering models that are fit, precise, sound and simple. Process discovery algorithms automatically obtain these models from event logs, records of past process executions, enabling insights into the underlying process. However, event logs often contain incomplete and infrequent behaviour, which presents significant challenges for these algorithms. To address these issues, this thesis proposes, a new process discovery technique called OptIMIIst. OptIMIIst guarantees soundness while handling both infrequent and incomplete behaviour and discovering locally optimal process trees. This technique, based on the Inductive Miner framework, operates in two steps. First, it creates candidate mining decisions for each process tree operator and then decides on the optimal decision through a local fitness and precision estimation. An experimental evaluation demonstrates that OptIMIIst produces high-quality process models and offers competitive fitness, precision, and simplicity compared to state-of-the-art techniques, while maintaining soundness.

Contents

List of Figures

List of Tables

Abbreviations

AIM Approximate Inductive Miner. 7, 8, 13, 15, 22, 26, 33, 36, 41–44, 46–48, 50, 52, 56

BIP Binary Integer Program. 23, 24, 28

BPIC Business Processing Intelligence Challenge. 38, 39, 42

BPM Business Process Management. 6, 9, 17

BPMN Business Process Model and Notation. 3, 10–12

DFG Directly Follows Graph. 3, 4, 10, 11, 13, 16, 21, 28–34, 36, 40, 50

EFG Eventually Follows Graph. 16, 27, 28, 36

ERP Enterprise Resource Planning. 9

ETM Evolutionary Tree Miner. 49

IFG Indirectly Follows Graph. 16, 36

ILP Integer Linear Program. 3, 8, 23, 24, 27–31, 33, 34, 36, 38, 44, 46–49, 52, 53

IM Inductive Miner. 3, 7, 8, 11, 12, 15, 19, 20, 22, 25–32, 37, 38, 40–43, 46, 47, 49, 50, 52, 53

IMc Inductive Miner - Incomplete. 7, 50

IMf Inductive Miner - Infrequent. 7, 8, 13, 42, 43, 50, 52, 55

LP Linear Program. 23

MAE Mean Absolute Error. 33, 34

MIP Mixed Integer Program. 23

PIM Probablisitc Inductive Miner. 7, 13, 39, 46, 47, 50

SMT Satisfiability modulo theories. 47, 48, 52

WF-net Workflow net. 11, 49

XES eXtensible Event Stream. 10

Chapter 1

Introduction

In many organizations, optimizing business processes has become essential for improving operational efficiency and customer satisfaction. Business processes are sequences of activities designed to achieve specific outcomes and deliver value to stakeholders [16]. These activities can vary in scope and detail, encompassing tasks performed manually by individuals, actions undertaken by external actors or automated operations executed by technical systems. Additionally, business processes play a vital role in organizing the resources utilized throughout these activities, ensuring that they are effectively allocated and managed. A typical example is an *order-to-cash* process, which encompasses the entire cycle from receiving and processing customer orders to managing inventory, invoicing, and collecting payment.

Business Process Management (BPM) is an approach to improving business processes. It includes the identification, design, execution, monitoring and continuous improvement of processes to enhance efficiency [16]. Identifying and designing processes, often using process models as visualizations, are challenging tasks that lay the foundation for further enhancements. Process mining, particularly automatic process discovery, has emerged as a supporting discipline in this context. Digital systems increasingly produce records of process executions, referred to as event logs. These logs enable process discovery techniques to automatically reconstruct process models from past executions. This data-driven approach provides a solid foundation for understanding process flows and guiding further analysis and improvements.

For a process model to be valuable for analysis, it needs to meet certain quality criteria. Four main quality dimensions are generally applied: fitness, precision, simplicity, and generalization [3]. Fitness measures evaluate how well the model captures the behaviour recorded in the event log. Precision measures assess how accurately the model represents only the behaviour contained in the log, avoiding enabling unseen behaviour in the model. This often puts the precision measure in direct competition with the generalization measure, which measures the model's ability to capture potential future unseen behaviour. Lastly, simplicity evaluates the readability of the model, often quantified as the size of the model, by counting the number of elements in it.

In addition to these four quantitative quality dimensions, soundness is another important model property. It ensures that process models are behaviourally correct and free from issues such as deadlocks or unreachable tasks. While the quality

dimensions focus on evaluating different aspects of the model's representation of the event log and its usability, soundness guarantees that the model can execute correctly. This binary property is often a prerequisite for model evaluation, reliable analysis and simulation [2].

Key challenges shared between many process discovery algorithms are data quality issues in event logs [29]. This includes infrequent and incomplete behaviour, which can significantly impact model quality. *Infrequent behaviour* refers to behaviour rarely observed in the log and may not represent the main stream process behaviour. Such behaviour is often the result of noise, i.e. recording or data entry errors, or rare exceptions in the process. *Incompleteness* refers to behaviour that is part of the process but is not captured in the log. This can easily happen in real-life event logs, as logs rarely capture all possible variations of a process, and an event log is just a historical except of the processes execution.

Many process discovery algorithms are particularly sensitive to noise, for example creating models that overfit the event log. Meaning a model becomes too tailored to the specific observed data, capturing noise or minor variations rather than the general behaviour of the process. As a result, these models fail to represent the core-process effectively. An effective discovery algorithm, therefore, must find a balance of avoiding noisy elements while at the same time handling incompleteness, as event logs rarely capture all possible process behaviour.

The Inductive Miner (IM) is a recursive process discovery algorithm that continuously decomposes the event log to build a process model. This decomposition is done by splitting the event log based on distinct activity sets and creating cuts based on specific operator definitions. As output the IM constructs a process tree, a specific type of process model that guarantees to be translatable into other sound process models [19]. The IM typically achieves high fitness scores in real-life event logs but does not balance the remaining quality metrics. This often leads to a loss in precision due to the so-called fallthrough methods used once no cut can be found. These fallthroughs sacrifice precision for fitness, making the IM very sensitive to infrequent and incomplete behaviour.

Simple extensions, like the Inductive Miner - Infrequent (IMf) [20] and Inductive Miner - Incomplete (IMc) [21] specifically target one of these problems each. IMf introduces a filtering technique to remove infrequent behaviour, while IMc applies heuristic cuts to find cuts. Other approaches, like the Approximate Inductive Miner (AIM) [14] and Probablisitc Inductive Miner (PIM) [9] attempt to address both issues by combining filtering and heuristic techniques. But a gap in process discovery techniques remain for an adaptive and exact method, which handles both infrequent and incomplete behaviour simultaneously. Ideally such a technique should adhere closely to the formal definitions of the IM framework while minimizing the amount of filtering applied.

This thesis aims to address the identified gap by introducing a new process discovery algorithm that guarantees soundness while tackling both infrequent and incomplete behaviour, all while closely adhering to the formal definitions of the IM framework and not relying on explicit filtering. This new technique, named Opt-IM-II-st (Optimization-InductiveMiner-Infrequent-Incomplete-eSTimation), ex tends the Inductive Miner (IM) framework by replacing the fallthrough step with a

two-step approach.

In the first step, the algorithm generates a set of cut candidates based on optimization problems representing relaxed versions of the IM cut definitions. In the second step, the best of these cut candidates needs to be chosen. This is done through a novel way to estimate local precision and fitness on incomplete process trees, selecting the cut candidate with the best estimate score. Through this two-step approach, OptIMIIst guarantees to mine locally optimal process trees according to the assumptions used in the relaxation of the cuts and estimation. Notably, the cut detection employed by OptIMIIst eliminates the need for explicit filter settings because the optimization can already perform implicit filtering. As a result, the algorithm has the potential to adapt to a wide range of real life event logs, with filtering clearly aimed at enabling optimal mining decisions, without requiring any user input.

As mentioned, OptIMIIst uses optimization problems to generate the cut candidates. Specifically, in this thesis, Integer Linear Program (ILP)s are employed to solve these problems. ILPs are well-suited for this task as they guarantee an optimal solution within defined constraints, ensuring that the discovered cuts are the best for a given operator.

This thesis will provide an implementation of the algorithm and evaluate the performance of the proposed miner compared to other state-of-the-art algorithms like AIM, IMf and the ILP-Miner. For the evaluation publicly available real-life event logs as well as on synthetic benchmark logs will be used.

This thesis is structured as follows: In Chapter 2, necessary background of the domain of process mining, process discovery and related topics will be sketched. In Chapter 3 specific concepts required for the main section will be introduced formally. It is followed by the main chapter of the thesis Chapter 4 in which OptIMIIst will formally be introduced. Following in Chapter 5 the practical implementation of OptIMIIst will briefly be discussed, which lays the groundwork for the evaluation of the developed method on both real life event logs as well as on data from the Process Discovery Contest. In Chapter 6 the results from the evaluation and contribution of this thesis is discussed and Chapter 7 discusses OptIMIIst place in the field of research, introducing related works. The thesis will be concluded in Chapter 8 with a final recap of the contents of this thesis and a final outlook on possible future paths of research and a closing conclusion.

Chapter 2

Background

This chapter informally introduces the background of the thesis. This includes explaining the role of business processes, event data, process models, conformance checking, and model quality.

Business processes are structured sets of activities or tasks designed to achieve specific organizational goals or outcomes. A popular example is an *order-to-cash* process, which starts with a company receiving an order of a product from a customer. In multiple steps, the company's different resources work on taking the product from a warehouse, packaging it, shipping it, creating an invoice and handling payment. The process ends successfully once the customer receives the order and the invoice is paid in full. However many errors and inefficiencies can occur throughout process execution. For example, goods may be delayed due to a staff shortage at a postal office, or payments may be postponed because of unclear instructions. Business Process Management is an approach to continuously improve business processes to mitigate errors and improve efficiency.

In today's digital landscape, most business processes, like the *order-to-cash* example, generate digital traces of their execution, which for example are found as data within IT systems such as Enterprise Resource Planning (ERP) systems. When scoped to a single process, this data is commonly referred to as an event log and can be analysed to gain insights into process performance and identify opportunities for improvement [33].

Process mining is a data-driven discipline combining data science and business process management techniques. By analysing event logs, process mining techniques enable insights into process execution patterns, bottlenecks, and inefficiencies. In doing so, process mining provides a factual basis for process improvement and automation. Key process mining tasks encompass process discovery, conformance checking, and enhancement. Process discovery involves constructing process models from event data. Conformance checking evaluates the alignment between actual executions and the models. Finally, enhancement uses the discovered models to guide data-driven process improvements.

Event logs are the primary data source for most process mining endeavours, including process discovery and conformance checking. They record the digital traces of past process executions, with each execution labelled with a unique *case id*. As such, for each *case id*, the log records the sequence of activities executed in the order

CaseID	Activity	Timestamp
Case 1	Activity A	10:00
Case 1	Activity B	11:00
Case 1	Activity D	12:00
Case 2	Activity A	11:00
Case 2	Activity C	12:00
Case 2	Activity D	13:00

Table 2.1: Example Log L_{ex1}

of their occurrence. Each entry in the event log needs to contain three essential attributes to be usable for process mining [33].

- **Case id**: A case identifier that groups events belonging to one instance of the process called a case. Common examples of case ids can be order or transaction ids.
- **Activity name**: The name of the activity or work performed. For example: 'Send Invoice' or 'Approve Order'.
- **Timestamp**: The time the activity was performed.

Besides these three required attributes, an event log can include additional attributes, such as the resource that executed the activity, the task duration or the cost associated with executing the activity [33]. However, for generality, most process discovery and conformance checking techniques are limited to these three standard attributes. Typically event logs are stored in either the general CSV format or more specialised standardized formats like the eXtensible Event Stream (XES) standard, which supports the extensible nature of event log attributes in an XML format [3].

Table 2.1 shows a small example event log L_{ex1} consisting of just two cases with three activities each and four distinct activities.

2.1 Process Models

Various modelling formalisms with differing levels of expressiveness and intended purposes have been developed over the years, ranging from simple transition systems and Directly Follows Graphs to more intricate representations like Business Process Model and Notation diagrams and Petri nets.

A Directly Follows Graph (DFG) is a simple graph-based representation of a process, illustrating the sequence of activities by showing which activities directly follow others. Each activity is represented as a node, while directed arcs between nodes indicate the order in which activities occur. DFGs often include designated start and end activities to clearly define the process's entry and exit points, and arc weights to represent the frequency of each directly follows relation within the log data [31]. An example of a DFG generated from L_{ex1} is depicted in Figure 2.1. DFGs can be constructed by replaying the log and adding missing nodes or arcs.

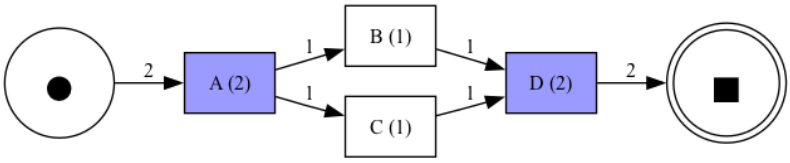


Figure 2.1: DFG of log L_{ex1}

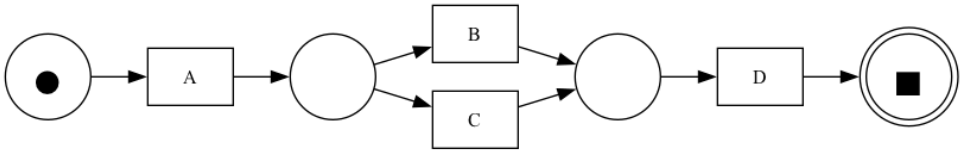


Figure 2.2: Petri net mined with the α-Miner on log L_{ex1}

While DFGs provide a straightforward overview of a process's general structure and are simple to construct DFGs cannot enforce control-flow constraints. This limits their suitability for more complex analyses that require a detailed understanding of process behaviour and structure and can even lead to wrong interpretations [31]. This is why formalisms like Petri nets and Business Process Model and Notation (BPMN) diagrams are often preferred. Although these models require more advanced mining algorithms to discover, they offer control-flow restrictions that enable meaningful replayability and more in-depth analysis of the process.

Petri nets, like the one shown in Figure 2.2 mined from log L_{ex1} using the α-Miner [1], are bipartite graphs. They consist of places, transitions, and directed arcs connecting them, forming a transition system. Places can hold tokens and transitions fire by consuming and producing tokens, thereby changing the state of the net. This state evolution defines the language of the model and as such the process it represents [26]. As previously mentioned, an important property of Petri nets is soundness, which ensures that the net is behaviourally correct [2].

A specific type of Petri nets are Workflow net (WF-net)s. WF-nets are designed to model workflows with a single designated start place and a single end place, ensuring a clear beginning and conclusion for each process instance. All other elements in a WF-net are positioned on paths from the start to the end, enforcing a well-defined flow of activities, without unnecessary or disturbing elements. Since these properties, like having a single start and end state[1] match the properties that are expected in process execution they are widely used in process mining for analysing and validating real-world workflows [2].

Figure 2.3 shows a BPMN diagram of log L_{ex1} mined by the IM [19]. Instead of using places to model control flow restrictions, a BPMN diagram uses gateways, such as exclusive (XOR) and parallel (AND) gateways, to define branching and synchronization points within the process. Beyond this basic control flow, BPMN supports a wide range of additional notations to capture various process aspects, including events, tasks, and subprocesses. Events in BPMN can represent occurrences that trigger actions or changes in the process flow, such as starting, intermediate,

[1]Which does not necessarily translate to having a single outcome. See start and end activity enhanced DFG

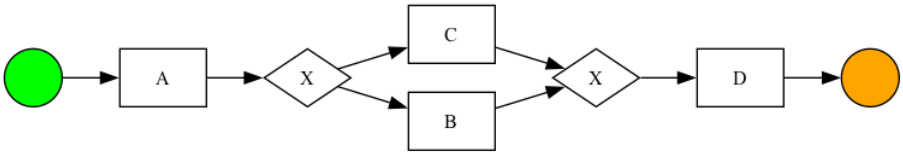


Figure 2.3: BPMN Diagram mined with the IM of log L_{ex1}

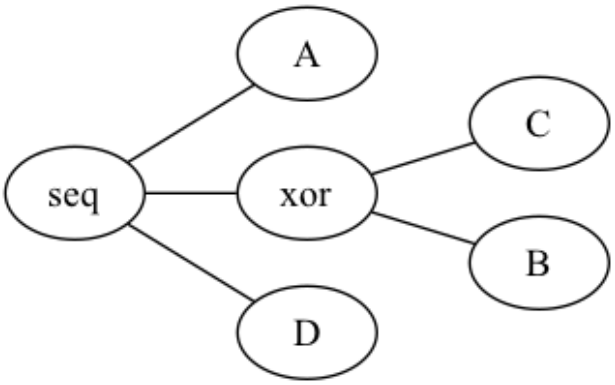


Figure 2.4: Process Tree $\mathcal{T}_{ex1}$ mined with the IM of log L_{ex1}

or ending conditions. Additionally, BPMN incorporates elements like message flows and resource pools to illustrate interactions between different participants or departments, which makes it suitable for modelling complex, multi-actor workflows. This richness in notation allows BPMN to model not only the sequence of tasks but also the context and dependencies within a process. It provides a comprehensive view that is accessible to both technical and business audiences. Furthermore, practical implementations of BPMN in tools like Camunda enable the automatic execution of these models, enhancing BPMN's utility by bridging the gap between process modelling and process automation.

However, most process discovery algorithms are restricted to modelling the basic control flow structure and lack support for discovering the full range of BPMN elements, such as message flows or complex event handling.

Another popular modelling formalism are Process Trees, like the one seen in Figure 2.4, mined by the IM [19]. Process Trees represent processes in a hierarchical tree structure where each node denotes an activity or operator. They are of particular interest because they can be guaranteed to be transformed into sound workflow nets and will be further discussed in Section 3.4.

2.2 Process Discovery

Process Discovery is a central task in process mining, dedicated to automatically generating a process model from a given event log. The primary objective is to reveal the underlying structure of the process solely from observed execution data, without relying on prior knowledge of the process or manual modelling. However, this task faces challenges like handling infrequent, noisy, and incomplete behaviour. Infrequent behaviour in event logs can stem from various causes negatively impact

the discovery algorithms performance. The causes include rare exceptions in the process or noise that introduce additional, missing, or incorrect events. Furthermore, when using historical data, there is no guarantee that all possible process behaviours are represented in the event log. A log may capture only a subset of the process's full range of activities, making the observations inherently incomplete.

Various automated process discovery techniques have been introduced, discovering process models based on an input event log. For a comprehensive overview, the author refers to [7], with a selection of related works outlined in Chapter 7.

For many discovery techniques, logs that contain infrequent or incomplete behaviour, are challenging to handle. Manual log cleaning to minimize the amount of infrequent or incomplete behaviour is often time-consuming and requires a certain level of insight into the underlying process. As a result, filtering has traditionally been an important preprocessing step to remove infrequent behaviour from an event log. Without including further process knowledge in the filtering approach, usually the most infrequent DFG arcs or activities are gradually removed. This type of filtering is either applied as data cleaning before process discovery or incrementally applied during the mining process, as seen in the AIM, IMf or PIM. In the former application filtering is either applied until a specific threshold is reached or, in the case of filtering as part of the filtering algorithm, until a mining decision is made. However, filtering comes with the downside of incurring a loss of information and thus reducing fitness.

2.3 Conformance Checking and Model Quality

Besides the already discussed soundness property, process discovery tries to balance four quality dimensions of the output model. These dimensions are fitness, precision, generalization and simplicity.

Fitness measures how well the model captures the behaviour observed in the log. For example, looking at the model in Figure 2.2 of L_{ex1}, the whole log is replayable on the model. The model has perfect fitness. If the log is changed by adding the trace $\langle \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D} \rangle$, this new trace is not be replayable on the model and decreases the fitness of the model on the log. Similarly, if the expressiveness of the model is decreased by, for example, making Activity B unplayable, fitness is also decreased since not all traces are replayable anymore. Fitness is typically quantified on a scale from 0 to 1, with different methods available for measuring it across various modelling formalisms. For Petri nets, three fitness measures are particularly popular in process mining [3]: footprint-based fitness, which compares the relationships between activities; token-replay fitness, which tracks token movements through the model penalizing remaining or missing tokens [28]; and alignment fitness, which matches log traces to the closest possible model traces, penalizing movements on model only or log only [4]. Each measure provides a different level of accuracy, with alignment fitness offering the most precise evaluation at a higher computational cost.

Precision measures how accurately a model represents only observed behaviour, without enabling additional behaviour not observed in the event log. Again, looking at Figure 2.2, exactly two traces are playable on the model. The same traces are also observed in the log, resembling perfect precision. If traces are removed from the

log, the precision decreases. Similar if more behaviour is enabled in the model, by creating a Petri net that allows the arbitrary execution of activities, the precision also decreases. Precision is also quantified on a 0 to 1 scale, with similar variations of the footprint, token-replay and alignment-based precision typically used.

Generalization measures how well the model enables future yet unseen behaviour or, more specifically, quantifies how likely it is that the model will support unseen cases. Generalization as such can be seen as the counterpart to precision. While precision aims to avoid underfitting, generalization aims to avoid overfitting the model on the observed behaviour. Some measures for generalization have been proposed in literature like [17].

Simplicity refers to how easily a process model can be understood, interpreted, and maintained. A model with high simplicity avoids unnecessary complexity, showing only the essential elements needed to capture the process behaviour. For example, instead of using the split and join only for Activities B and C in Figure 2.2, a less simple model could also split Activities A and D into separate paths duplicating the activities. Such a model has the same fitness, precision and generality as the one in Figure 2.2, but is be more complicated. Overly complex models can be difficult to analyse, and may lead to decreased usability in practice. Simplicity is often balanced with fitness, precision, and generalization, as an overly simplified model may miss important details, while an overly complex one may become cumbersome without significantly improving accuracy. Simplicity is typically assessed through structural metrics, such as the number of nodes and arcs, or by examining whether the model captures core behaviours without redundant constructs.

Chapter 3

Preliminaries

This chapter presents the formal definitions necessary for the thesis. It begins with general process mining notation in Section 3.1, followed by the notation and definitions for activity relations and graphs in Section 3.2, Petri nets in Section 3.3 and process trees in Section 3.4. Section 3.5 introduces the IM algorithm, and Section 3.6 introduces concepts of the AIM, that will be reused for OptIMIIst. The chapter concludes with an overview of the formalization and definitions related to linear programming.

3.1 General Notation

In this thesis, event logs are denoted by L, while individual logs are marked by subscription such as L_1, L_2. Each event log comprises traces, denoted by σ, which are sequences of activities ordered by occurrence. Symbols represent activities within a trace so that a trace appears as $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$. For this thesis, exact timestamps are not considered. Instead, only the sequence of activities is relevant. For example, the trace $\langle a, b \rangle$ represents a trace consisting of two activities, where a is executed first and b afterwards. $|L|$ denotes the number of traces in the log and $||L||$ the total number of activities.

To denote traces that contain no activities, an empty trace is denoted by ϵ. V denotes the set of unique traces, in terms of distinct activity sequences in the log. They are distinguished by their event log of origin using subscripted identifiers such as V_1, V_2. $|V|$ represents the number of unique traces in the log. Similarly, Σ denotes the alphabet of unique activities in the log and $|\Sigma|$ is the number of unique activities in the log. Σ_1, Σ_2 indicating their event log of origin.

The set of activities that appear at the beginning of any trace in an event log is denoted as $\textsc{Start}(L)$, while those at the end are denoted as $\textsc{End}(L)$. Additionally, the extended functions $\textsc{Start}(L, a)$ and $\textsc{End}(L, a)$ represent the number of times activity a appears as a start or end activity, respectively, within the log.

	A	B	C	D
A	0	1	1	0
B	0	0	0	1
C	0	0	0	1
D	0	0	0	0

Table 3.1: Matrix representation of the DFG of L_{ex1}

3.2 Activity Relations and Graphs

Between two activities exists a follows relationship if there exists a trace in the log in which one activity is followed by the other. As such the directly-follows relationship $a \nrightarrow b$ holds for L if $\exists_{\sigma \in L} \sigma = \langle \ldots, a, b, \ldots \rangle$. Furthermore, $|a \nrightarrow b|$ denotes the number of times the activity a is directly followed by b in L. Similarly, the eventually follows relationship $a \nrightarrow^+ b$ holds if $\exists_{\sigma \in L} \sigma = \langle \ldots, a, \ldots, b, \ldots \rangle \vee a \nrightarrow b$. Representing that activity, b is executed at some point after activity a in a trace of the event log L, but not necessarily immediately. The notation $|a \nrightarrow^+ b|$ indicates the number of times a is eventually followed by b and behaves similarly to the directly-follows relationship. Finally the strict indirectly follows relationship $a \nrightarrow^* b$ holds for L if $\exists_{\sigma \in L} \sigma = \langle \ldots, a, \ldots, b, \ldots \rangle$. So if activity a is followed by b at some point in the log but not immediately.

From the directly follows activity relationship, the DFG can be constructed. The DFG is a directed graph representation of a log where each activity is represented as a node and arcs exist between two nodes if there is a directly-follows relationship between the corresponding activities. In the weighted version of the DFG, the arcs are weighted by the frequency of the directly-follows relationship occurring in the log, and the DFG is commonly extended by start and end activities. Besides the graph representation like seen in Figure 2.1 for log L_{ex1} a matrix representation like in Table 3.1 commonly used in implementations. The Eventually Follows Graph (EFG) and Indirectly Follows Graph (IFG) are constructed analogously.

3.3 Petri nets

Petri nets are bipartite graphs consisting of two types of nodes: places (denoted as P) and transitions (denoted as T), which are interconnected through directed arcs (denoted as F)and are formally a tuple $PN = (P, T, F)$ with [26]:

- P being the set of places of the Petri net
- T being the set of transitions, which are labelled by the activity it executes, or the silent activity τ representing the absence of an activity.
- F being the set of directed arcs representing the flow between places and transitions with $F \subseteq (P \times T) \cup (T \times P)$

The state of a Petri net, a marking, represents the current distribution of tokens across its places. A marking transitions to a new marking as transitions are executed. A transition $t \in T$ is enabled when all its input places (those connected to

it by incoming arcs) contain at least one token. Once executed, the transition consumes tokens from its input places and produces tokens in its output places (those connected to it by outgoing arcs), creating the following marking.

The sequences of transitions that can be fired, starting from an initial marking, define the language of the Petri net, which encompasses all possible executions allowed by the model. This language provides a formal foundation for evaluating process behaviours and their alignment with observed data in event logs.

An optional property of Petri nets is soundness, which ensures the net is behaviourally correct. For a Petri net to be considered sound, it needs to satisfy three key conditions [2]:

- **Option to complete:** From any reachable marking, it is always possible to eventually reach the final state, ensuring that the process terminates appropriately.
- **Proper termination:** Upon reaching the final state, no other transitions are enabled, ensuring the process ends without remaining active tasks.
- **No dead transitions:** Every transition in the net is potentially fireable in at least one execution sequence, meaning no transitions are never used.

These properties collectively ensure that the process modelled by the Petri net is free from deadlocks, unnecessary tasks, and incomplete executions.

3.4 Process Trees

Section 2.1 introduced various process models, their applications in process mining and BPM and the concept of replaying a log on a model. Process trees have been mentioned briefly and an example of a process tree representing the process from L_{ex1} can be seen in Figure 2.4. Process trees represent the flow of a process in a hierarchical tree structure with, leaf nodes being labelled with an element from $\Sigma \cup \{\tau\}$. Elements from Σ denote the execution of the specific activity and τ representing unobserved state changes. Inner nodes, in contrast, are labelled with an operator from the set of operators $\oplus \in \{\rightarrow, \times, ||, \circlearrowleft\}$, each of which defining specific execution rules for its child subtrees.

For simplicity's sake, this thesis focuses exclusively on binary process trees, meaning each operator node has exactly two children. Since any process tree has an language equivalent binary process tree this can be done without loss of generality [18]. Process Trees in this thesis are denoted with $\mathcal{T}$, with their language being represented as $\mathcal{L}(\mathcal{T})$. Process trees can also be expressed in a linearized form by unrolling their structure, as in the example $\mathcal{T}_{ex1} = \rightarrow (A, \times(B, C), D)$.

Each operator introduces specific rules on the execution of the tree, thereby defining the language of a process tree. The language definitions for the various operators are as follows:

- **Leaf nodes:** The leaf nodes represent either the execution of an activity or the lack thereof, giving us the base cases:

$$\mathcal{L}(a) = \{\langle a \rangle\} \text{ for } a \in L \quad \text{and} \quad \mathcal{L}(\tau) = \{\epsilon\}.$$

- **Sequence operator** ($\rightarrow$): The sequence operator requires that the left (or upper) child of the operator node be executed before the right (or lower) child. This is expressed as the concatenation of the languages of the subtrees:

$$\mathcal{L}(\rightarrow(\mathcal{T}_1, \mathcal{T}_2)) = \{\sigma_1\sigma_2 \ : \forall_{\sigma_1 \in \mathcal{L}(\mathcal{T}_1), \sigma_2 \in \mathcal{L}(\mathcal{T}_2)}\}.$$

- **Exclusive choice operator** ($\times$): The exclusive choice operator allows for only one of the child subtrees to be executed:

$$\mathcal{L}(\times(\mathcal{T}_1, \mathcal{T}_2)) = \{\sigma : \forall_{\sigma \in (\mathcal{L}(\mathcal{T}_1) \cup \mathcal{L}(\mathcal{T}_2))}\}.$$

- **Parallel operator** ($\|$): The parallel operator allows for the independent parallel execution of both child subtrees, resulting in the interleaved language of the subtrees:

$$\mathcal{L}(\| \ (\mathcal{T}_1, \mathcal{T}_2)) = \{\sigma : \forall_{\sigma_1 \in \mathcal{L}(\mathcal{T}_1), \sigma_2 \in \mathcal{L}(\mathcal{T}_2)}, \sigma \text{ is an interleaving of } \sigma_1 \text{ and } \sigma_2\}$$

- **Loop operator** ($\circlearrowleft$): The loop operator requires the execution of the left subtree (the body part) and allows for arbitrary repetition of the body part after executing the right subtree (the redo part):

$$\mathcal{L}(\circlearrowleft \ (\mathcal{T}_1, \mathcal{T}_2)) = \mathcal{L}(\mathcal{T}_1)(\mathcal{L}(\mathcal{T}_2)\mathcal{L}(\mathcal{T}_1))^*.$$

Two common special structures found in process trees are so called τ-skips and τ-loops. Both utilizing τ as one of their subtrees, the τ-skip is an exclusive choice node with one τ-leaf and one non τ-subtree enabling optional behaviour. τ-loops on the other hand consist of an loop operator with the redo part being a τ leaf allowing for the arbitrary repetition of the body part.

A special feature of process trees is the guarantee that they can be transformed into sound workflow nets. By using the transformation rules shown in Figure 3.1, a sound Petri net can be created. Specifically, for each type of process tree construct, corresponding rules are applied:

- **Normal Events**: A single activity leaf is transformed into a Petri net, consisting of a single transition, labelled with the activity name. The transition is connected to one input and one output place.

- **Sequence**: For a sequence operator the subtrees are chained by their input and output places enforcing the sequential execution.

- **Parallel**: For a parallel operator a τ transition is incoming connected with the input place and outgoing connected with the two input places of the subcomponents. The same holds for the output places of the subcomponents which are joined through a τ transition to the output place.

- **Exclusive Choice**: For an exclusive choice operator an input place is connected to the subcomponents, either directly like in the example in Figure 3.1 or with τ transitions moving a token to the input place of the subcomponent. This creates a split-and-merge structure.

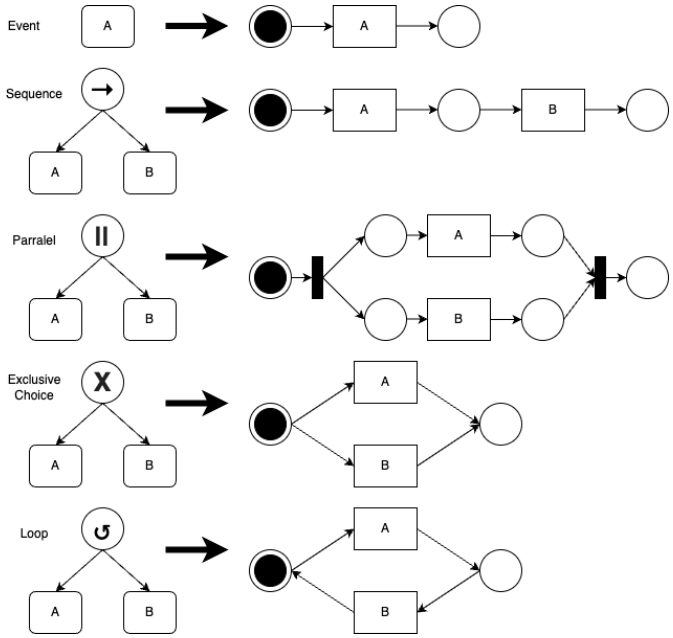


Figure 3.1: Transformation rules to create Petri nets from process trees

- **Loop Cut**: For a loop operator, the input place is connected to the input place of the body via a τ transition, or directly like in Figure 3.1. The output place of the body is linked to both the input place of the repetition and the output place of the loop using τ transitions, creating a structure where tokens can either exit the loop or re-enter the body via the redo part.

These transformation rules ensure that the resulting Petri net is sound, adhering to properties such as proper termination and deadlock-freeness.

3.5 Inductive Miner

The IM framework is a process discovery algorithm which recursively constructs process trees. By iteratively splitting the event log, based on its activity set, into smaller sublogs the IM builds a process tree as it refines each sublog.

Since this thesis focuses on binary process trees, a specific version of the IM, referred to as the IM for binary cuts, is introduced. As mentioned in Section 3.4, this choice comes without loss of generality. Algorithm 1 contains the pseudocode of the IM with binary cuts.

The algorithm first attempts to identify a base case through the BASECASE(L) function, which aligns with the base cases from the process tree language definition. These base cases cover instances where the event log L contains either a single activity or empty traces. In such cases, the algorithm returns a process tree consisting of a single leaf node, with $\mathcal{T} = a$ for a single activity or $\mathcal{T} = \tau$ for empty traces.

If no base case is found, the algorithm proceeds by attempting to identify a cut using the FINDCUT(L) function. If a cut is successfully identified, the SPLITLOG(L) function partitions the log, and the recursion continues on the resulting sublogs. In

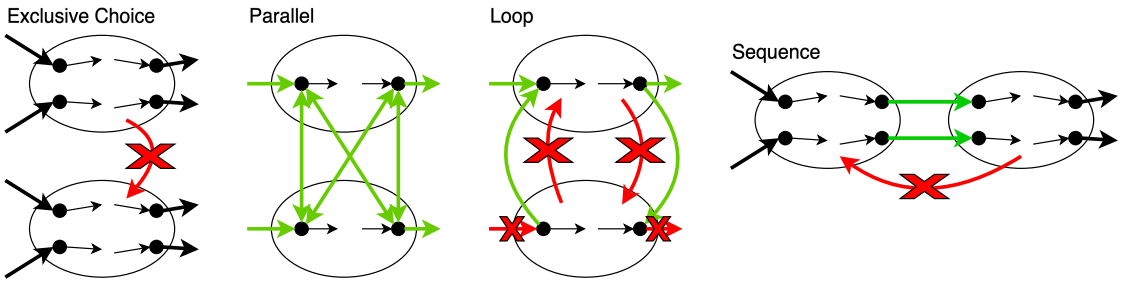


Figure 3.2: Visual representation of IM cuts

the absence of both a base case and a suitable cut, the IM resorts to predefined fallthrough mechanisms.

Algorithm 1 Inductive Miner from [19]

```
function IM(L)
    bc ← BASECASE(L)
    if bc ≠ ∅ then
        return bc
    end if
    ⊕, Σ1, Σ2 ←FINDCUT(L)
    if ⊕ ≠ ∅ then
        L1, L2 ←SPLITLOG(L, ⊕, Σ1, Σ2)
        return ⊕(IM(L1),IM(L2))
    else
        return FALLTHROUGH(L)
    end if
end function
```

FindCut Function

If no base case applies, the IM proceeds with the FINDCUT(L) function to find an appropriate partition of activities along with a process tree operator. For a partitioning to be admissible a set of definitions, unique to each operator and based on activity relationships, must be satisfied. The definition for the general case are detailed in [19], the for this thesis relevant, binary versions are outlined in the following providing a visual representation in Figure 3.2.

Sequence Cut The definition of the sequence cuts requires that the eventually follows relationships between activities in the two partitions are strictly one-directional and exhaustive. That is, every activity in partition A must eventually be followed by every activity in partition B, with no reverse relationships from B to A. This requirement enforces clean sequential behaviour and is formalized in equation (3.1).

$$\forall_{a \in \Sigma_0, b \in \Sigma_1} a \nrightarrow^{+} b \wedge b \nrightarrow^{+} a \tag{3.1}$$

Exclusive Choice Cut The exclusive choice cut definition requires that no directly follows relationships exist between activities in different partitions. This enforces that, within any trace of that operator node, only activities from one partition are executed, ensuring exclusivity. In the visual representation of a DFG exclusive choice cuts are easily detectable by finding two independent sub-graphs with no connecting edges. This is formalized by disallowing any relationships between any pair of activity a from partition one and any activity b from partition two, as shown in (3.2).

$$\forall_{a \in \Sigma_0, b \in \Sigma_1} a \nrightarrow b \wedge b \nrightarrow a \tag{3.2}$$

Parallel Cut The parallel cut is defined by two partitions that are fully interconnected in a unidirectional manner, meaning each activity in partition A is directly followed by all activities in partition B, and vice versa. Additionally both partitions must include at least one start activity and one end activity. This allows all activities including start and end activities to be executed independently in parallel. This is encapsulated in the two equations in (3.3) and (3.4).

$$\forall_i \Sigma_i \cap \textsc{Start}(L) \neq \emptyset \wedge \Sigma_i \cap \textsc{End}(L) \neq \emptyset \tag{3.3}$$

$$\forall_{a \in \Sigma_0, b \in \Sigma_1} a \twoheadrightarrow b \wedge b \twoheadrightarrow a \tag{3.4}$$

Loop Cut The loop cut is composed of two named partitions: the body part, which represents the primary sequence of activities, and the redo part, which execution allows for repetitions of the body. For a loop cut to be valid, all start and end activities must belong to the body partition. Additionally, any arcs in the DFG that transition from the body partition to the redo partition must originate from start activities in the body. Likewise, arcs from the redo partition that re-enter the body must terminate at start activities. These constraints ensure that the loop structure is correctly represented, allowing the body to repeat following the redo activities. These constraints are encompassed in the definition by (3.5) for the location of the start and end activities and (3.6) and (3.7) for the flow restrictions.

$$\Sigma_0 \supseteq \textsc{Start}(L) \cup \textsc{End}(L) \tag{3.5}$$

$$\forall_{a \in \Sigma_0} \exists_{b \in \Sigma_1} b \twoheadrightarrow a \Rightarrow a \in \textsc{Start}(L) \tag{3.6}$$

$$\forall_{a \in \Sigma_0} \exists_{b \in \Sigma_1} a \twoheadrightarrow b \Rightarrow a \in \textsc{End}(L) \tag{3.7}$$

Split Log Function

The SplitLog function is responsible for dividing the event log into two sub-logs based on the identified cuts. For sequence, parallel, and exclusive choice cuts, the function creates two sub-logs by filtering each trace to retain only the activities belonging to the respective partitions of the cut.

For loop cuts, when a loop structure is detected, by the trace including an activity of the body part being followed by an activity of the redo part, SplitLog creates new traces for each separate loop iteration. Each iteration trace is assigned a unique case identifier to avoid conflicts with the existing case identifiers within the sub-logs. For the τ-loops the SplitLog function will detected loops once an end activity of the log is followed by a start activity of the log and create new traces accordingly.

Fallthrough Functions

The IM is not a complete algorithm when limited to base cases and cuts. It is trivial to construct examples in which the algorithm will be unable to find a cut or base case to proceed with. In those cases IM makes use of a set of FALLTHROUGH functions, applying tree structures that are usually fitness preserving but sacrifice precision [18]. Typical inclusions in the set of fallthroughs are τ-skips, τ-loops and as a last resort a FlowerModel, a process tree that allows for arbitrary execution of the remaining activities, with log precision preserving techniques being preferable [18]. A FlowerModel has the structure $\circlearrowleft (\times(a_1, \cdots, a_n), \tau)$ allowing for arbitrary execution of activities looped by the loop operator.

3.6 Approximate Inductive Miner

AIM is a process discovery algorithm based on the IM framework. Similarly to OptIMIIst it introduces a novel fallthrough mechanism, using clustering techniques to discover cut candidates for each operator and cut scoring to select the cut to continue the recursion with. For a detailed description of AIM the author refers to [14]. For the thesis the base case extension of AIM, to handle an increased amount of empty traces, is particular relevant.

Instead of using the default IM base cases, AIM also enables the use of τ-skips and τ-loops as potential base cases and can be selected if the log only contains a single type of activity and empty traces. If the log only contains empty traces a τ leaf is returned. In any other case three scores are calculated, $S_\times$ for the $\times(A, \tau)$, $S_\circlearrowleft$ for the $\circlearrowleft (A, \tau)$ and S_a for a single activity leaf. The scores are calculated as such:

$$S_\times = \frac{|[\sigma \in L : \sigma = \epsilon]|}{|L|} \tag{3.8}$$

$$S_\circlearrowleft = \frac{|[\sigma \in L : |\sigma| > 1]|}{|L|} \tag{3.9}$$

$$S_a = \frac{|[\sigma \in L : |\sigma| = 1]|}{|L|} \tag{3.10}$$

AIM applies the base case with the highest score.

3.7 Linear Programming

Linear Programming, also known as Linear Optimization, is a mathematical method used to find the global optimum of a given optimization problem. In order to maintain linearity and as such convexity of the solution space, these models are restricted to linear relationships between decision variables. The standard form of

a Linear Programming problem is expressed as follows:

$$\begin{aligned} \text{minimize} \quad & c^T x && (3.11)\\ \text{subject to} \quad & Ax \leq b && (3.12)\\ & x \geq \mathbb{Z}^p \times \mathbb{R}^q && (3.13) \end{aligned}$$

With $n = p + q$ variables $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. The objective function (3.11) determines the optimization direction of the problem and can be either a minimization or maximization problem. The optimization is constraint by a set of constraints like constraints (3.12) do in a matrix form. Generally in Linear Programming greater than equal and less than equal signs are used with equals being an extension realized as a combination of the two. Lastly in (3.13) the decision variables are defined, which if not further restricted, can be freely chosen to solve the Linear Programming Problem within the restricted solution space.

From this general standard form specialised problem classes can be derived. Linear Program (LP) are programs with $p = 0$ and as such have purely continuous variables. They can be solved to optimality in polynomial time complexity by the use of algorithms like the Simplex algorithm [12] or the interior-point method [15]. This changes when adding restrictions to the variable space. ILPs are models with $p = 0$, while programs with both $p \geq 1$ and $q \geq 1$ are considered Mixed Integer Program (MIP)s. Both ILPs as well as MIPs are harder to solve having an non-polynomial computational complexity. This extends to Binary Integer Program (BIP) with $x \in \{0, 1\}$ as a subclass of ILPs. Known algorithms like the Branch&Bound Algorithm [23], which utilizes a decision tree and solves increasingly restricted LPs through the Simplex method, do not solve these problems faster than exponential worst case time complexity.

Because of its guarantee to find the optimal solution to the given problem and expressive modelling capabilities Linear Programming is used in a wide variety of domains like logistics, economics and also has prior use in process mining for example in the ILP-Miner [35]. In practice solvers like Gurobi or SCIP can be employed. The offer modeling and solver capabilities for LP, ILP, MIP, BIP in different programming languages.

Cutting Planes and Symmetry Breaking To speed up the solving process of ILPs, MIPs, or BIPs, cutting planes or symmetry-breaking constraints can be used. As an example, consider the following BIP, which resembles a Knapsack Problem with items a, b, c and a capacity of 3.5. The valid optimal integer solutions to this problem are $\{a = 1, b = 0, c = 1\}$ and $\{a = 0, b = 1, c = 1\}$.

$$\begin{aligned} \text{maximize} \quad & a + b + 5c && (3.14)\\ \text{subject to} \quad & a + b + 2c \leq 3.5 && (3.15)\\ & a, b, c \in \{0, 1\} && (3.16) \end{aligned}$$

Cutting planes are additional optional constraints that remove non-integer solutions from the solution space. For the example, the constraint $a + b + c \leq 2$ does not remove any valid integer solutions, but it does remove the non-integer solutions

$\{a = 1, b = 0.5, c = 1\}$ and $\{a = 0.5, b = 1, c = 1\}$. By removing these non-integer solutions, they will not be discovered as potentially better solutions, this can speed up the solver without impacting the discovered optimal solution.

Symmetry-breaking constraints, on the contrary, are designed to avoid multiple equivalent solutions [25]. The example BIP has two equivalent solutions, both with an objective value of six. Adding the constraint $a \geq b$ would remove the solution $\{a = 0, b = 1, c = 1\}$ from the solution space without affecting the maximal objective value to be discovered, since $\{a = 1, b = 0, c = 1\}$ is a symmetric equivalent solution. By removing symmetric solutions from the solution space, the solver does not have to keep rediscovering multiple equivalently good solutions.

While cutting planes and symmetry-breaking constraints can have a positive impact on solver performance, this is not guaranteed. There is no guarantee that the constraint will impact the solving of a specific problem instance and, on the contrary, can also have a detrimental effect on performance. This is because solution times for ILPs scale not only with the number of variables but also with the number of constraints.

Chapter 4

Opt-IM-II-st

In this section, the OptIMIIst process discovery algorithm is introduced. As an extension of the IM framework, it offers the structural guarantee of soundness of its output process model. OptIMIIst employs an adaptable approach to handling incomplete and infrequent behaviour without the need for manual input or filtering.

As its basis, OptIMIIst proposes an advanced cut detection based on optimization problems and a local cut comparison score based on fitness and precision estimators. In combination, these approaches are used to find a locally optimal cut. The cut detection creates a set of cut candidates, while the comparison score is used to select the most suitable one.

In Algorithm 1, the IM for binary cuts was introduced. t makes use of fallthrough functions in cases in which the FindCut function can not find a base case or cut. OptIMIIst replaces this set of fallthrough functions with a single subroutine, always returning a locally optimal cut without the need for fallthroughs, like the flower model.

The pseudocode of the OptIMIIst fallthrough can be found in Algorithm 2 and a visualization in Figure 4.1. It operates in two main steps corresponding to this method's two main contributions. In the first step, a set of candidate cuts using the $\text{FindCuts}_{\text{OptIMIIst}}$ function is created. This set is created by formulating and solving operator-specific optimization problems to find the locally optimal binary activity partitioning for each operator under the constraint that neither partition is empty. The τ-skip and τ-loop are manually added to the candidate set. These are two particularly useful structures covering behaviour non-empty binary cuts can not adequately cover. Since it is necessary for further evaluation of the cut candidates,

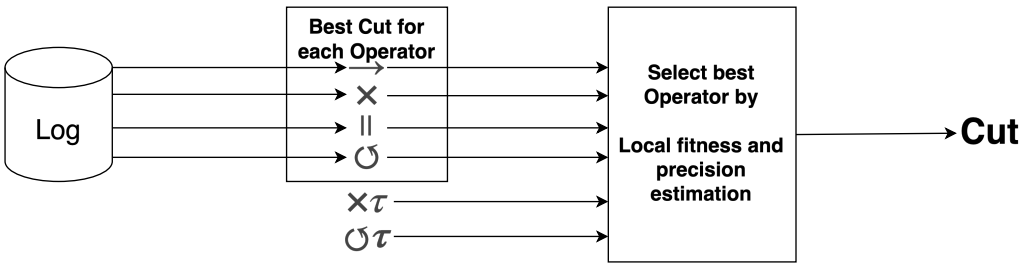


Figure 4.1: Overview picture of the OptIMIIst fallthrough

the log is split during the FindCuts$_{\text{OptIMIIst}}$ routine based on a slightly adjusted SplitLog function. To decide which of the cut candidates to continue the recursion with, the EvaluateCut function calculates a quality measure. Selecting the cut with the highest quality measure and continuing the recursion on the corresponding sublogs. The measures are based on local fitness and precision estimation and are unique to the operator used. A slight change is also made to the base cases of the algorithm. The BaseCase function is extended by enabling τ-skips and τ-loops using the same scores as AIM in Section 3.6. This is done because OptIMIIst's imperfect cuts potentially create a larger number of empty traces in its sub logs than the base IM,

Algorithm 2 Opt-IM-II-st Fallthrough

1: **function** FALLTHROUGH$_{\text{OptIMIIst}}(L)$
2: $C \leftarrow \{\text{FindCuts}_{\text{OptIMIIst}}(L)\} \cup \{(\times, L, \emptyset), (\circlearrowleft, L, \emptyset)\}$
3: $(\oplus_{res}, L_{res1}, L_{res2}) \leftarrow \underset{(\oplus, L_1, L_2) \in C}{\text{argmax}}$ EVALUATECUT$((\oplus, L_1, L_2))$
4: **return** $\oplus_{res}(\text{IM}(L_{res1}), \text{IM}(L_{res2}))$
5: **end function**

In the following in Section 4.1, the changes to the base cases are covered, which correspond to the changes made in [14]. In Section 4.2, the FINDCUTS$_{\text{OptIMIIst}}$ function is discussed, going over each operator and its corresponding optimization problem, and in Section 4.3 the cut evaluation is presented, introducing the fitness and precision estimation for each operator.

4.1 BaseCase

Making cuts that do not conform to the cut definitions of the IM framework as outlined in Section 3.5 will potentially create a larger amount of empty traces in the sublogs. For example applying a hypothetical sequence cut $\rightarrow (\{A\}, \{B, C\})$ to the event log $L_B = \{\langle A, C\rangle \langle B, C\rangle\}$ will create the sublogs $L_{B1} = \{\langle A\rangle, \epsilon\}$ and $L_{B2} = \{\langle C\rangle, \langle B, C\rangle\}$. One way to handle these additional empty traces and sequences of the same activity is to always use a τ-skip or τ-loop when applicable. This will lead to high fitness, enabling this behaviour at every step, but also to a decline in precision and a larger, more complicated model. Because OptIMIIst aims to balance the quality dimension, OptIMIIst uses the same BaseCase extension as AIM outlined in Section 3.6. Scoring τ-skip, τ-loop and a single activity base case and selecting the one with the highest score.

4.2 FindCuts

As described previously, the FINDCUTS$_{\text{OptIMIIst}}$ function returns locally optimal partitions for each of the four operators $\{\rightarrow, \times, ||, \circlearrowleft\}$. It is used in cases where the IM can not find a base case or a cut that conforms to the formal definitions from Section 3.5.

FINDCUTS$_{\text{OPTIMIIST}}$ finds binary cuts on relaxed versions of the formal IM cut definitions. In this context, relaxed means that the new equations allow for minimal deviations from the original equations. By using this relaxation inside an optimization problem, maximizing conforming and minimizing deviating behaviour, the algorithm attempts to find an equivalent decision to the *maximal non-trivial* cuts of the IM. To formulate and solve the optimization problems ILPs have been chosen because they natively offer the optimization aspect of the problem.

The problems are formulated in separate ILPs, unique to each operator, with a few shared structures are used in all of them. To determine in which partition the activity resides, the set of binary variables $x_a \forall a \in \Sigma$ is used. To give a direction, required for the sequence and loop operator, it is defined that $x_a = 1$ if the activity is in the left partition, also called partition one, of the process tree and $x_a = 0$ if the activity is in the right partition, also called partition two. Since only non-trivial cuts are valuable, the partitions are required to be non-empty. This is realized through the two constraints (4.6) and (4.7), present in every ILP. Additionally, at times, logical exclusive-OR operations, like $z_{a,b} = x_a \veebar x_b$, are required between binary variables. These are usually realized through variants of the four constraints (4.1), (4.2), (4.3) and (4.4). In cases in which the exclusive-OR bound variable has a neutral or negative effect on the objective value, the constraints (4.3) and (4.4) can be dropped without affecting the optimal solution.

$$\veebar \geq x - y \tag{4.1}$$
$$\veebar \geq y - x \tag{4.2}$$
$$\veebar \leq 2 - x - y \tag{4.3}$$
$$\veebar \leq x + y \tag{4.4}$$

It is important to note that a constructed ILP will always return the best possible cut for its assigned operator, even in cases where the event log's structure favours a different operator entirely. The objective of the optimization problem and its ILP implementation is not necessarily to yield a "good" cut for the process model as a whole, but to provide the best achievable cut for the specified operator.

4.2.1 Sequence Cut

The sequence cut, as formally defined in (3.1), aims to identify global sequential behaviour within the event log. The definition requirement is that only EFG arcs from partition one to partition two are allowed and must be complete, while any arcs from partition two to partition one are prohibited.

In the EFG example shown in Figure 4.2, the IM will not only not find a sequence cut but no valid cut at all. However, the cut $\rightarrow (\{A\}, \{B, C\})$ is the sequence cut which preserves the most behaviour, with only a single arc with little arc weight not conforming the formal cut definition. While manually identifying cuts works for a small example like this, it is not suitable for large real-life event logs. As such, an automated way of detecting cuts is required, with the goal of preserving as much behaviour as possible that fits the formal definitions.

For the sequence cut, the definition of the optimization problem is relaxed by the formulation not requiring EFG arcs from each activity from partition one to

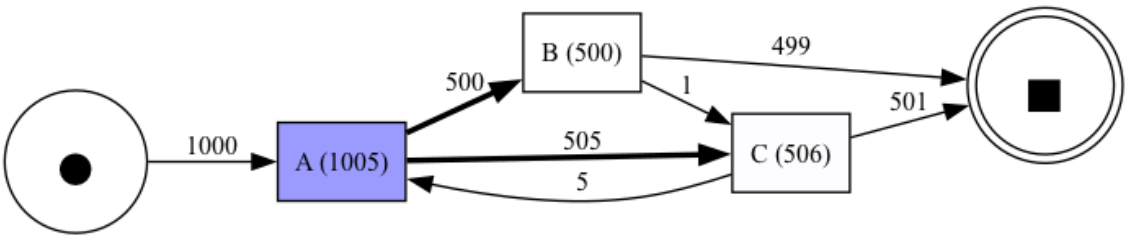


Figure 4.2: DFG of a log in which the IM will not find a cut, but $\rightarrow (\{A\}, \{B, C\})$ is suitable.

each activity in partition two. Instead, it maximizes the arc weights in the allowed direction, from partition one to partition two. Secondly the formulation does not enforce the lack of arcs in the opposite direction but minimizes them. This relaxation not only stays near the formal definition, but encapsulates the intuition to preserve as much original behaviour as possible. Encapsulating this into a single optimization problem statement: The problem to solve is to find a partitioning that maximizes the arc weights in the correct direction while minimizing those in the opposite direction.

The following ILP implements this optimization problem by maximizing an objective function defined in (4.5). This function maximizes the sum of conforming arc weights while penalizing non-conforming ones. Specifically, for each pair a and b in Σ, the term $|a \twoheadrightarrow^+ b| \cdot (x_a - x_b)$ represents the arc weight $|a \twoheadrightarrow^+ b|$, multiplied by $(x_a - x_b)$. This multiplier evaluates to 0 when activities a and b are in the same partition, meaning it does not influence the objective value. It evaluates to 1 if a and b are in different partitions according to the conforming direction, contributing positively to the objective value. Conversely, it evaluates to -1 when the direction does not conform, which negatively impacts the objective value. Constraints (4.6) and (4.7) are the non-empty constraints ensuring neither partition is empty. The only decision variable of this ILP is the binary x_a variables, making this linear program a BIP.

$$\text{maximize} \quad \sum_{a \in \Sigma} \sum_{b \in \Sigma} |a \twoheadrightarrow^+ b| \cdot (x_a - x_b) \tag{4.5}$$

$$\text{subject to} \quad \sum_{a \in \Sigma} x_a \geq 1 \tag{4.6}$$

$$\sum_{a \in \Sigma} (1 - x_a) \geq 1 \tag{4.7}$$

$$x_a \in \{0, 1\} \qquad \forall a \in \Sigma \tag{4.8}$$

Notably the optimization problem will also find a definition conforming sequence cut in the EFG if one exists. This is due to the fact that no sequence cut can have a better objective value than a definition conforming sequence cut.

4.2.2 Exclusive Choice Cut

The definition of the exclusive choice cut (3.2) states that no arcs are allowed between the partitions in the DFG. Again, considering the DFG in Figure 4.3, the IM will not

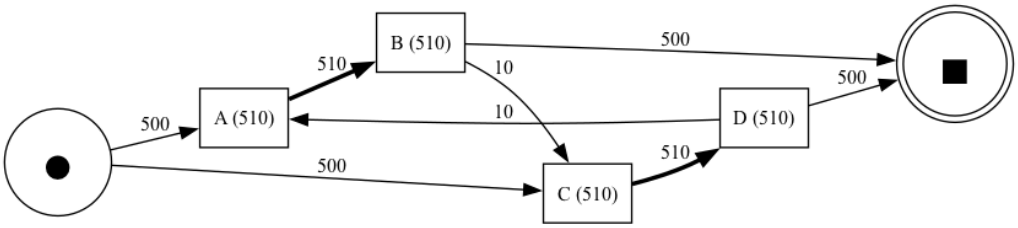


Figure 4.3: DFG in which the IM will not find a cut, but $\times(\{A, B\}, \{C, D\})$ is suitable

find a cut. But the cut preserving the most behaviour possible is $\times(\{A, B\}, \{C, D\})$. This cut only removes two arcs that break the definition, both of which have low arc weights compared to the global behaviour.

By relaxing the formal definition for the optimization problem, minimizing arcs between partitions serves as a continuous measure instead of the binary condition of the original requirement. To achieve this in an ILP an additional binary auxiliary variable $z_{a,b}$ is require for each pair of activities $a, b \in \Sigma$. This variable is used to determine if the activities are in the same partition or not and is defined as the XOR of the activities $z_{a,b} = x_a \veebar x_b$. Resulting in the objective function in (4.9).

To ensure correct assignment of $z_{a,b}$, the constraints (4.10) and (4.11) are used as defined previously. The third constraint of the XOR construct is omitted because the ILP is a minimization problem. Breaking the XOR construct by setting $z_{a,b} = 1$ would have a neutral or negative effect on the objective.

$$\begin{aligned}
\text{minimize} \quad & \sum_{a \in \Sigma} \sum_{b \in \Sigma} |a \twoheadrightarrow b| \cdot z_{a,b} && && (4.9)\\
\text{subject to} \quad & (4.6) \wedge (4.7) \wedge (4.8) && && \\
& z_{a,b} \geq x_a - x_b && \forall a, b \in \Sigma && (4.10)\\
& z_{a,b} \geq x_b - x_a && \forall a, b \in \Sigma && (4.11)\\
& z_{a,b} \in \{0, 1\} && \forall a, b \in \Sigma && (4.12)
\end{aligned}$$

This cut will not find the maximal non trivial binary cut in a situation, where multiple possible binary exclusive choice cuts can be found in an event log. Instead, the ILP will return the first definition-conforming exclusive choice cut it encounters in its solution process. Any definition-conforming exclusive choice cut will also be an optimal solution to the ILP. An imperfect cut cannot achieve an objective value of zero or lower, since at least one arc in an imperfect cut breaks the definition, thereby increasing the objective value, making it worse than a definition-conforming cut. This limitation is not a concern when used as a fallback, as no perfect cut can exist once a fallback scenario is reached.

An, in this thesis unused, extended version of the ILP includes a symmetry-breaking part of the objective function like:

$$\sum_{a \in \Sigma} \sum_{b \in \Sigma} |a \twoheadrightarrow b| \cdot z_a + \frac{|\sum_{a \in \Sigma} x_a - \frac{|\Sigma|}{2}|}{|\Sigma|}$$

Preferring a cut that balances the number of activities in either partition. It applies

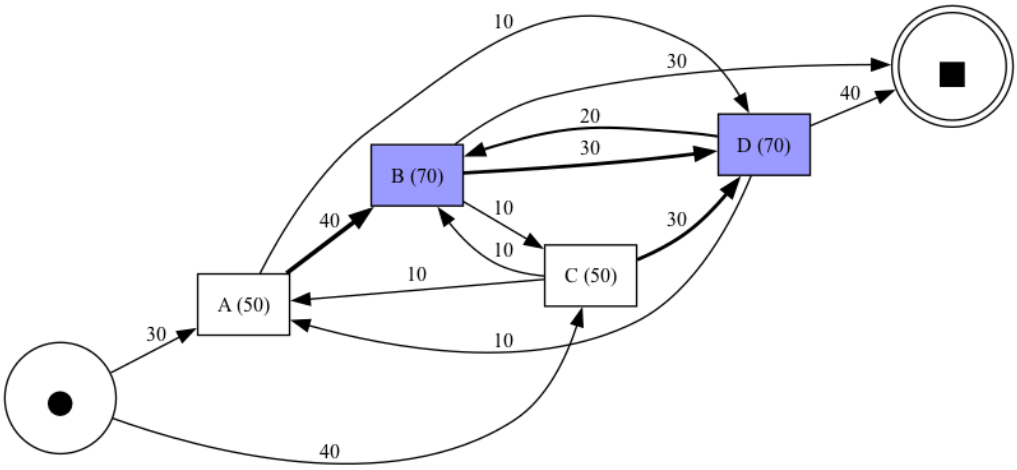


Figure 4.4: DFG in which the IM will not find a cut, but $||(\{A, B\}, \{C, D\})$ is suitable

in cases in which two or more solutions have the same objective value, leading to the more maximal cuts to be chosen if multiple cuts have the same objective value.

Another symmetry-breaking constraint that can be added without impacting the output is assigning a single activity to one of the partitions from the start. This is possible because an exclusive choice cut $\times(\mathcal{T}_1, \mathcal{T}_2)$ is equivalent to its mirrored version $\times(\mathcal{T}_2, \mathcal{T}_1)$, always creating two equivalent solutions of the ILP. Adding the single constraint (4.13) would assign the first activity of the alphabet randomly to partition one, breaking this symmetry.

$$x_{\Sigma[0]} = 0 \tag{4.13}$$

4.2.3 Parallel Cut

Looking at the example DFG in Figure 4.4, no valid cut can be found, but on closer inspection only a single arc, specifically an arc between $A \twoheadrightarrow B$, is missing for the cut $||(\{A, B\}, \{C, D\})$ to be valid. Intuitively, for a parallel cut, the most interconnected components, with the largest sum of arc weights between them, are likely to form the partitions that constitute the parallel cut.

Since the definition in (3.3) and (3.4) is twofold, requiring the partitions to be completely interconnected and at the same time contain start and end activities in both of the partitions, the relaxation of this definition is twofold. The first part of the relaxation allows for the absence of start or end activities in a partition but encourages a balance between the number of start and end activities in each partition. The second part maximizes the DFG arc weights of arcs crossing between the partitions, relaxing the complete interconnectivity requirement of the activities in the DFG.

The suitability of the first part of this relaxation can be defended by considering two long-running subprocesses that run in parallel within an organization. It would be uncommon for an end activity from subprocess A to be directly followed by a start activity from subprocess B, which would break the formal definition of a parallel

cut. However, the subprocesses still run in parallel, with activities between them creating an almost interconnected structure.

Implementing this in an ILP requires some auxiliary variables. As such $z_{a,b}$ is used in the same way as in the exclusive choice cut, but now with the full XOR definition including the constraints (4.15) and (4.16). Additionally, we need the imbalance of start and end activities, which is collected in a new variable $b \in \mathbb{N}$. The constraint, (4.17) b is set to the absolute imbalance of start and end activities. b is subsequently subtracted from the objective value in (4.14). The value is otherwise calculated by summing the total arc weights between the partitions through the variables $z_{a,b}$, similar to the exclusive choice cut.

In contrast to the exclusive choice cut, the objective function (4.14) is maximized instead of minimized. It now maximizes the arc weights between the partitions using $z_{a,b}$ as a multiplier and reducing the objective value by the imbalance of start and end activities.

$$
\begin{aligned}
\text{maximize} \quad & \sum_{a \in \Sigma} \sum_{b \in \Sigma} z_{a,b} \cdot |a \twoheadrightarrow b| - b && (4.14)\\
\text{subject to} \quad & (4.6) \wedge (4.7) \wedge (4.8) \wedge (4.10) \wedge (4.11) \wedge (4.12) && \\
& z_{a,b} \leq x_a + x_b \quad \forall a, b \in \Sigma && (4.15)\\
& z_{a,b} \leq 2 - x_a - x_b \quad \forall a, b \in \Sigma && (4.16)\\
& b \geq |\Sigma_{a \in \Sigma} \textsc{End}(L, a) - 2 \cdot \Sigma_{a \in \Sigma} x_a \cdot \textsc{End}(L, a)| && (4.17)\\
& + |\Sigma_{a \in \Sigma} \textsc{Start}(L, a) - 2 \cdot \Sigma_{a \in \Sigma} x_a \cdot \textsc{Start}(L, a)| && \\
& b \in \mathbb{N} && (4.18)
\end{aligned}
$$

This ILP could be extended in the same way as the exclusive choice cut to avoid symmetry of the solution by adding the constraint (4.13). This is possible because the process trees $||(\mathcal{T}_1, \mathcal{T}_2)$ and $||(\mathcal{T}_2, \mathcal{T}_1)$ are again equivalent.

4.2.4 Loop Cut

IMs definition of the loop consists of three parts as defined in (3.5), (3.6) and (3.7). All start and end activities must be part of the body part (Partition One) and are not allowed in the redo part (Partition Two). Additionally, if DFG edges from partition two lead to partition one, they have to lead to start activities. The other way around DFG edges that originate from partition one and lead to partition two need to originate from the end activities of the log.

When looking at Figure 4.5, the IM will again not find a valid cut, but the cut $\circlearrowleft (\{A, B, C\}, \{D, E\})$ preserves the main stream process behaviour. The loop behaviour can easily be detected by the frequent arcs between Activity C and D and E and A. Only infrequent behaviour like the single occurrence of E as a start activity and the shortcut D to B break the loop behaviour.

The relaxation used is, to encourage general loop behaviour, discouraging breaking behaviour, like shortcuts or start and end activities in the redo part. When encompassing this in the relaxation and formulating it into an optimization problem the different concerns from the relaxation need to be split. The main goal of the optimization problem is to maximize the loop behaviour. To implement this a large

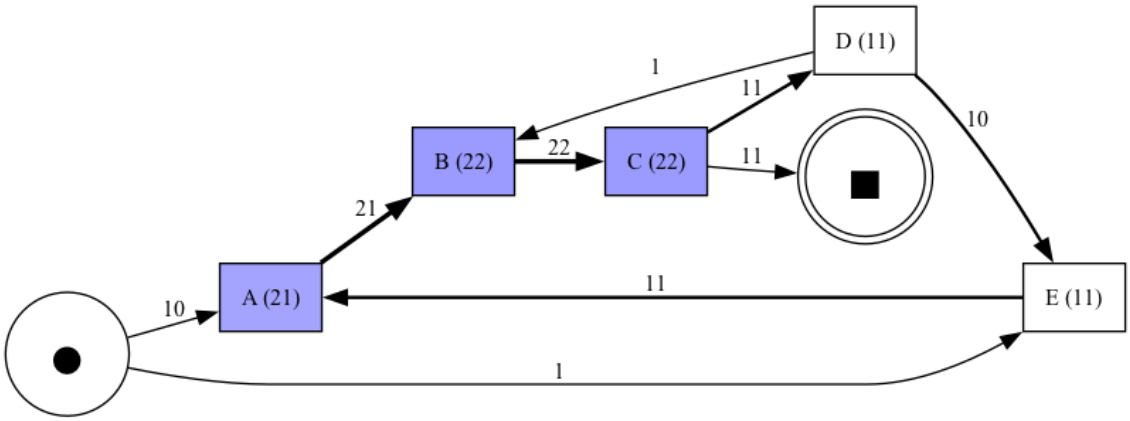


Figure 4.5: DFG of a log in which the IM will not find a cut, but $\circlearrowleft(\{A, B, C\}, \{D, E\})$ is suitable.

set of auxiliary variables is required. The binary variables $v_{a,b}$ and $w_{a,b} \in \{0, 1\}$ for each activity tuple symbolize if an arc between the activities will be an arc conforming the cut definition. $v_{a,b}$ requires a to be a start activity and $w_{a,b}$ b to be an end activity. Setting the variables is realized through constraints (4.20) to (4.29) and makes use of additional auxiliary variables rs_a and re_a symbolizing if an activity a is a start or end activity of the redo loop. $\mathcal{X}$ is used as the indicator function in some of the constraints.

To finally encapsulate this global behaviour in the objective function (4.19) the arc weights for arc in which $v_{a,b}$ and $w_{a,b}$ are summed up and arcs for which the auxiliary variable $f_{a,b}$ is one are subtracted. $f_{a,b}$ is one for the activity combinations from different partitions which are not one for either $v_{a,b}$ or $w_{a,b}$.

To add the first part of the definition, the objective value is additionally discounted by the number of start and end activities in the redo partition.

$$\begin{aligned}
\text{maximize} \quad & \sum_{a\in\Sigma}\sum_{b\in\Sigma}(v_{a,b}\cdot|a \twoheadrightarrow b| + w_{a,b}\cdot|a \twoheadrightarrow b|) && (4.19)\\
& - \sum_{a\in\text{START}(L)\cup\text{END}(L)}(1-x_a)\\
& - \sum_{a\in\Sigma}\sum_{b\in\Sigma} f_{a,b}\cdot|a \twoheadrightarrow b|\\
\text{subject to} \quad & 4.6 \wedge 4.7 \wedge 4.8\\
& \text{rs}_a \leq (1-x_a) && \forall a\in\Sigma \quad (4.20)\\
& \text{re}_a \leq (1-x_a) && \forall a\in\Sigma \quad (4.21)\\
& \text{rs}_b \geq (x_a - x_b)\cdot a \twoheadrightarrow b && \forall a\in\text{END}(L) b\in\Sigma \quad (4.22)\\
& \text{re}_a \geq (x_b - x_a)\cdot a \twoheadrightarrow b && \forall b\in\text{START}(L) a\in\Sigma \quad (4.23)\\
& w_{a,b} \geq \text{rs}_b + \mathcal{X}_{\text{END}(L,a)} - 1 && \forall a,b\in\Sigma \quad (4.24)\\
& w_{a,b} \leq \text{rs}_b && \forall a,b\in\Sigma \quad (4.25)\\
& w_{a,b} \leq \mathcal{X}_{\text{END}(L,a)} && \forall a,b\in\Sigma \quad (4.26)
\end{aligned}$$

$$v_{a,b} \geq \mathrm{re}_a + \mathcal{X}_{\mathrm{START}(L,b)} - 1 \qquad \forall a, b \in \Sigma \quad (4.27)$$
$$v_{a,b} \leq \mathrm{re}_a \qquad \forall a, b \in \Sigma \quad (4.28)$$
$$v_{a,b} \leq \mathcal{X}_{\mathrm{START}(L,b)} \qquad \forall a, b \in \Sigma \quad (4.29)$$
$$f_{a,b} \geq |(x_a - x_b)| \cdot a \twoheadrightarrow b - w_{a,b} - v_{a,b} \qquad \forall a, b \in \Sigma \quad (4.30)$$
$$f_{a,b}, v_{a,b}, w_{a,b} \in \{0, 1\} \qquad \forall a, b \in \Sigma \quad (4.31)$$
$$\mathrm{rs}_a, \mathrm{re}_a \in \{0, 1\} \qquad \forall a \in \Sigma \quad (4.32)$$

4.3 Cut Selection by local Quality Estimation

After running the FindCuts$_{\text{OptIMIIst}}$ function, the potential cuts that are generated represent the best possible cuts for their respective operators according to the relaxation embedded in the ILPs. Multiple competing cuts might fit well for some logs, such as the one represented as an DFG in Figure 4.3. For the example at hand, the sequence cut ILP will deliver the cut $\rightarrow (\{A, C\}, \{B, D\})$, the exclusive choice cut ILP will return $\times(\{A, B\}, \{C, D\})$ and the parallel cut ILP will return $||(\{A, B\}, \{C, D\})$. All cuts that intuitively could potentially fit the log as a next mining decision. Only the loop cut ILP will produce a partitioning that is not suitable, which is not problematic because there are valid alternatives.

The proposed selection method is to evaluate the quality of a cut at the local level. Like for the candidate cut detection the goal is to be close to formal definitions. As such fitness and precision estimates for each operator are created. To select the final mining decision the fitness and precision estimates are aggregated into a single score by using the F_1 score:

$$F_1 = 2 * \frac{\text{Fitness} \cdot \text{Precision}}{\text{Fitness} + \text{Precision}}$$

Because the τ-loop and τ-skip are particularly useful structures enabling arbitrary repetition of behaviour and optional behaviour, they are added to the candidate set. Proxy measures from the AIM [14] are used to estimate the quality of these two structures.

In the following the estimators for each operator will be introduced. A key part of precision estimation will be to compare probability distribution, between observed behaviour in the log and expected behaviour in the log projection. For this the Mean Absolute Error (MAE) of continuous probability sets of size n is used:

$$\mathrm{MAE}(P_{\text{Observed}}, P_{\text{Expected}}) = \frac{1}{|P_{\text{Expected}}|} \sum_{i=1}^{n} |P_{\text{Observed}_i} - P_{\text{Expected}_i}|$$

4.3.1 Sequence Cut

Starting with the *sequence cut* shown in Figure 4.6, the ILP defined in Section 4.2 will find the potential sequence cut $\rightarrow (\{A, B\}, \{C, D\})$. To estimate the fitness of this cut, the algorithm evaluates the proportion of DFG arc weights that are not replayable after applying the cut. This proportion is then compared to the total

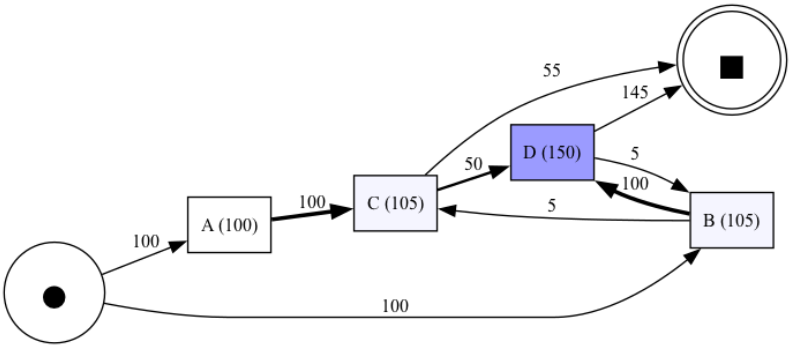


Figure 4.6: DFG in which the sequence ILP will find the cut $\rightarrow (\{A, B\}, \{C, D\})$

weights of arcs. Consequently, the fitness of a sequence cut is defined as the inverse of this proportion, resulting in the following fitness formula:

$$F_{seq} = 1 - \frac{\Sigma_{a \in \Sigma_2} \Sigma_{b \in \Sigma_1} |a \nrightarrow b|}{\Sigma_{a \in \Sigma} \Sigma_{b \in \Sigma} |a \nrightarrow b|} \tag{4.33}$$

For the precision estimation, it is assumed that after applying the cut the probability $P_{Expected}$, of transitioning from any $a \in \textsc{End}(L_1)$ to any $b \in \textsc{Start}(L_2)$, to be uniform. This assumption is made because the cut removes any possible causal dependencies between activities in different partitions. Activity frequencies are ignored when calculating these transition probabilities in the model, as they are properties of the log rather than of the model. As a result, the transition probability is calculated as $\frac{1}{|\textsc{Start}(L_2)|}$. We compare these expected probabilities with the observed probabilities in the log $P_{Observed}$.

This intuition can be applied to the cut $\rightarrow (\{A, B\}, \{C, D\})$ and the corresponding DFG in Figure 4.6. In the observed data, Activity B is more likely to be followed by Activity D than by Activity C, while similarly Activity A is exclusively followed by Activity C but never by Activity D. After applying the proposed sequence cut the model supports sequences $A \rightarrow D$, a direct loss of precision enabling unseen behaviour. Also the information that the sequence $B \rightarrow C$ was less likely than the sequence $B \rightarrow D$ is lost, removing this causality from the model.

As introduced in the introduction of this section, the MAE is used to compare these probabilities, which leads to a precision formula:

$$P_{seq} = 1 - \text{MAE}(P_{Observed}, P_{Expected}) \tag{4.34}$$

4.3.2 Exclusive Choice Cut

Continuing with the *exclusive choice cut*, fitness is calculated as the fraction of existing DFG arcs to the total possible arcs between activities from different partitions. This approach measures how well the model restricts behaviour between these partitions, aligning with the expected behaviour of an exclusive choice cut where the partitions should ideally have no connections to the others. As such, the resulting fitness estimation formula is defined as:

$$F_{xor} = 1 - \frac{\Sigma_{a \in \Sigma_2} \Sigma_{b \in \Sigma_1} a \nrightarrow b + \Sigma_{a \in \Sigma_1} \Sigma_{b \in \Sigma_2} a \nrightarrow b}{2 \cdot |\Sigma_1| \cdot |\Sigma_2|} \tag{4.35}$$

For the exclusive choice cut, precision is always 1, as it measures the model's ability to restrict behaviour to what is observed in the event log. In this cut, each choice follows distinct paths, preventing unintended transitions between partitions. When the cut is applied, no additional paths are introduced. Conversely, as discussed in Section 4.2, any imperfect exclusive choice cut will prevent replay of certain behaviours across partitions, effectively restricting behaviour. These conditions result in a perfect precision score.

$$P_{xor} = 1 \tag{4.36}$$

4.3.3 Parallel Cut

Similar to the exclusive choice cuts precision, the fitness of a *parallel cut* is also always 1, as the parallel branches can independently execute and, as such, enable all possible interleaved combinations of traces from the sub-logs.

$$F_{and} = 1 \tag{4.37}$$

On the other hand, the precision estimation needs to consider that, applying a parallel cut potentially enables a lot of behaviour by interleaving all possible traces of the sublogs. First $\textsc{avglen}(\mathcal{V}_1)$ and $\textsc{avglen}(\mathcal{V}_2)$ are the average variant lengths of the sub-logs $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively. With these the expected number of variants is calculated as $V_{\text{exp}} = 2^{\textsc{avglen}(\mathcal{V}_1)+\textsc{avglen}(\mathcal{V}_2)}$. This calculation reflects the potential number of distinct interleavings of activities from the two sub-logs, assuming that any trace from $\mathcal{V}_1$ can be paired with any trace from $\mathcal{V}_2$ to form a unique variant. This measure is then compared to the observed number of variants from the original log before splitting, $|\mathcal{V}|$. The resulting precision formula is:

$$P_{and} = \frac{|\mathcal{V}|}{V_{\text{exp}}} \tag{4.38}$$

4.3.4 Loop Cut

For fitness of the loop cut, the focus is again on the behaviour that will no longer be replayable once the cut is applied. In this context, a key concern is the start and end activities that are part of the redo section of the cut. Once the cut is applied, these will become unavailable to be played as start or end activities. To quantify this impact, we compare the number of start and end activities present in the redo part to the total number of start and end activities in the original event log. The resulting fitness estimation formula is:

$$F_{loop} = 1 - \frac{|\textsc{Start}(L_2)| + |\textsc{End}(L_2)|}{|\textsc{Start}(L)| + |\textsc{End}(L)|} \tag{4.39}$$

Note that the relaxed version of the loop cut will not produce any violating edges after the cut is applied, making them unavailable for fitness estimation. Since the log split will create new subtraces in the body part, arcs originating in the redo part and crossing to the body part, without ending in original start activities, will just create new start activities in the body part. These new start activities will be handled in subsequent recursion steps.

For precision a similar notion as for the sequence cut is used. Instead of just using connections between any $a \in \text{END}(L_1)$ to any $b \in \text{START}(L_2)$, two transition probabilities are calculated and compared. $P_{obsA \to B}$ is the transition probability from any $a \in \text{END}(L_1)$ to any $b \in \text{START}(L_2)$ and $P_{obsB \to A}$ is the transition probability from any $a \in \text{END}(L_2)$ to any $a \in \text{START}(L_1)$. These transitions are compared to the uniform expected probability distribution. The intuition behind this process is the same as discussed for the sequence cut but doubled for each of the directions. Resulting in the formula:

$$P_{loop} = 1 - \frac{\text{MAE}(P_{obsA \to B}, P_{expA \to B}) + \text{MAE}(P_{obsB \to A}, P_{expB \to A})}{2} \tag{4.40}$$

4.4 Complexity

To create a notion of a worst-case runtime of OptIMIIst, the steps of OptIMIIst are evaluated separately and later combined into the final $\mathcal{O}$ notation. Since for most evaluations the DFG, EFG and IFG are required, it is assumed these are created on each recursion step a single time. To create the graphs one sweep through the log extending the graphs trace by trace is required, resulting in a runtime of $\mathcal{O}(||L||)$.

The SplitLog function has a worst-case runtime of $\mathcal{O}(||L||)$ requiring a single trace by trace sweep through the log to detect loops. Similarly the base case extensions adapted from AIM have a runtime of $\mathcal{O}(||L||)$ [14]. The FindCut function, however, has a potentially exponential runtime due to the use of ILPs for the optimization. While the fitness and precision estimation can be done in linear time to the log size $\mathcal{O}(||L||)$, the speed to solve a ILP scales expectationally to the number of variables and constraints used. The largest possible ILP and, as such, dominating ILP is the one for the loop cut. It has three sets of variables that exist for each combination of two activities and four sets of variables that exist for each activity. In terms of constraints, the ILP has seven constraints for each combination of activities and four for each activity. This results in the approximate complexity of $\mathcal{O}(2^{(3|\Sigma|+4|\Sigma|^2)+(7|\Sigma|^2+4|\Sigma|)})$ which can be simplified to

$$\mathcal{O}(2^{11|\Sigma|^2+4x})$$

All other cuts have smaller ILPs and, as such, lower complexity, leading to them being dominated by the larger loop cut term. Combining all this into a single term yields the $\mathcal{O}$ notation for the whole OptIMIIst fallthrough:

$$\mathcal{O}(2^{11|\Sigma|^2+4|\Sigma|} + ||L|| + ||L||) + ||L||)$$

The term can be simplified to.

$$\mathcal{O}(2^{11|\Sigma|^2+} + 3||L||)$$

The standard IM with its original set of fallthroughs has a worst-case runtime of $\mathcal{O}(|L| \cdot |\Sigma|^5)$ [18], which will be strictly better than applying an OptIMIIst fallthrough. As such, for the worst-case runtime, it is assumed that an OptIMI-Ist fallthrough is applied, at each step of the recursion. Since no two consecutive τ-skips or τ-loops can be applied the amount of recursion steps is limited to $|\Sigma|$ leading to the overall worst time complexity of:

$$\mathcal{O}((2^{11|\Sigma|^2+4|\Sigma|} + 3||L||) \cdot |\Sigma|)$$

Chapter 5

Evaluation

In this section, OptIMIIst is evaluated twofold. For this purpose, an implementation of OptIMIIst is used, with its details being covered in Section 5.1. First, synthetic evaluation data from the Process Discovery Contest 2024 is used to evaluate the algorithm's performance on specific structures and log issues to determine structural strengths and weaknesses in Section 5.2. Secondly, to evaluate OptIMIIst's performance on real-life event data, the publicly available Business Processing Intelligence Challenge (BPIC) datasets are used. OptIMIIsts implementation is compared to other state-of-the-art discovery algorithms in Section 5.3. All tests were conducted on macOS 14.5 with an M3-Max processor and 96GB RAM. The RAM usage consistently remained below 8GB throughout the whole evaluation, during both model discovery and conformance checking for all experiments performed.

5.1 Implementation

OptIMIIst is implemented in Python using Gurobi[1] and its direct Python bindings GurobiPy[2] to formulate and solve ILPs. The implementation follows the proposed pseudocode in Algorithm 2. For general functionality like log handling and the original cut detection of IM, the pm4py framework [3] has been used. The implementation is available on GitHub[4].

The ILPs from Section 4.2 have been implemented with the exact constraints and variables as shown there. There are exceptions for constraint (4.17) and (4.29). Constraint (4.29) requires an additional auxiliary variable and corresponding constraints to handle the absolute value and make it implementable. Similarly, constraint (4.17) also requires additional constraints to resolve the absolute values. None of the proposed symmetry-breaking constraints or cutting planes have been added to the evaluated version of the miner. The author suggests a runtime evaluation of these additions in future works.

Using a solver agnostic implementation or at least the support of open source

[1] https://www.gurobi.com
[2] https://pypi.org/project/gurobipy/
[3] https://github.com/pm4py
[4] https://github.com/Lalaluka/optimiist

solvers like SCIP[5] is considered desirable. However, it would not impact the algorithm's performance outside of its runtime and, as such, is not covered in this thesis.

5.2 Process Discovery Contest Data

The Process Discovery Contest is a since 2016 annually held competition for process discovery techniques co-located at the International Conference on Process Mining. In contrast to the BPIC, the Process Discovery Contest focuses on the technical ability of process discovery algorithms to adapt to common issues and structures encountered in event logs.

As such, the Process Discovery Contest data set includes configurable synthetic data to objectively evaluate the performance of discovery algorithms on different structures and log imperfections. The goal of the analysis of OptIMIIst on this dataset is to determine specific structural weaknesses and strengths of the algorithm by looking at them in isolation and combination.

The most recent Process Discovery Contest 2024 data[6] was used for this thesis. The competition was won by a version of the proprietary Pyrrhus algorithm with an aggregate score of 90.86%; the audience award was won by the second placed also proprietary PIM with a score of 89,46%[7].

5.2.1 Setup

The Process Discovery Contest 2024 dataset includes 288 event logs, with 96 sets of event logs being used to evaluate and discover the models. Each set consisting of training, base, test and ground truth log. Models are discovered by the algorithm under evaluation on the training log. Afterward, every trace of the test and base log is tested regarding alignment-based fitness against the model. If the test trace fits the model better than the base trace, it is classified as a positive, and if the base trace fits the model better than the test trace, it is classified as a negative. This classification is compared to the ground truth classification attached to the ground truth traces as trace attributes; as such, a classification of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) is created. From these values, the negative and positive precision and recall are calculated through the formulas:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

From these values the positive and negative scores are calculated by using the F_1-Score. These again are aggregated into a single F_1-Score calculated as:

$$F_1 = \frac{\text{TP} * \text{P}_{Score} + \text{TN} * \text{N}_{Score}}{\text{TP} + \text{TN}}$$

[5]https://www.scipopt.org/

[6]PDC24 data: https://data.4tu.nl/datasets/3cfcdbb7-c909-4f60-8bec-62c780598047/1

[7]PDC24: https://icpmconference.org/2024/process-discovery-contest

The final score of the algorithm is calculated as the average F_1-Score over all 288 logs.

The training logs are organized by eight configurable settings, six of which concern the structure of the ground truth model the logs are generated from, and two concern the log quality. Starting with the former six, the long-term dependencies (LD) attribute determines whenether the model includes long-term dependencies; setting it turns bypasses of dependent activities on or off. The second attribute disables or enables the inclusion of optional choice constructs (OR); the third attribute concerns routing constructs (RC), meaning the inclusion of invisible tasks; the fourth concerns the inclusion of optional tasks (OT); and the fifth, the inclusion of duplicate Tasks (DT), leading to the relabelling of tasks with labels already included in the model. These first five settings are either enabled or disabled in the following, symbolized with a (-0) for disabled and (-1) for enabled after their abbreviation. Besides these settings, the seventh setting handles the inclusion of loops (LO) and has three settings: disabling loops completely (-0), enabling simple loops (-1), and complex loops (-2). Simple and complex loops are differentiated in their well-definedness, with simple loops having clearly separate main and loop flows. In contrast, complex loops can include shortcuts between loop and main flow, similar to the formal and relaxed definitions of the loop cut in the IM compared to OptIMIIsts relaxation.

Besides these model-based settings, the remaining two settings concern changes done to the log. The noise setting can either be enabled (N-1) or disabled (N-0), activation leads to every trace with probability 20% to contain issues, either one random event is removed (40%), moved (20%), or copied (40%). Additionally, one additional switch enables 20 positive and 20 negative traces to be pre-labeled, if they are positive or negative traces, this can be disabled (PL-0) or enabled (PL-1). It should be noted that OptIMIIst does not use this parameter, so no performance increase on this attribute should be expected.

5.2.2 Results

OptIMIIst achieved an overall score of 86,19% throughout the evaluation, which would have placed it in third place at the process discovery contest. The results clearly show that OptIMIIst can handle a wide range of model and log challenges, but it has weaknesses in handling long-term dependencies and duplicated tasks.

The detailed results can be observed in Figure 5.1. One outlier is the trace with the settings LD-0, LO-1, OR-1, RC-0, OT-1, DT-1, N-0, and PL-0, which only archives a score of 21%, resulting in this single outlier showing up multiple times as an outlier in the chart. When inspecting the DFG of this specific log a very convoluted unstructured middle part of the process can be identified, while the end and start are more sequential. When comparing this with the Petri net OptIMIIst produced, the start and end of the process are similarly modelled. But OptIMIIst decided on a loop cut including transition "t25" as the redo part early during mining. This activity is seemingly heavily effected by the duplicate tasks setting, creating an unfitting and also imprecise model.

The worst average performance can be detected through the introduction of

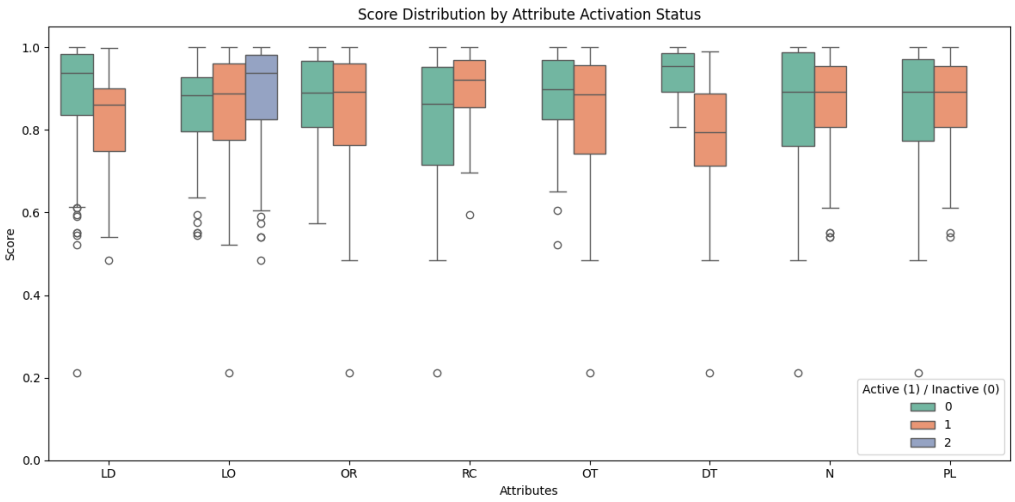


Figure 5.1: Performance of OptIMIIst on the PDC 2024 Dataset

duplicated activities, reducing the average score by 14%, with long-term dependencies in second place inducing a 6% drop in average score. The worse performance on logs with long-term dependencies matches the observation noted about AIM in [14]. Interestingly enough, introducing loops did not hurt the miner's performance. Loops were a weakness previously noted about AIM in [14]. Similarly to the AIM, OptIMIIst handled noise quite well with no drop in average score by introducing noise.

The drop in performance when introducing duplicated activities can be attributed to the core principle the IM applies. Namely splitting logs through their activities. The IM and, in its extension OptIMIIst, has no way to split two activities that are labelled the same into different partitions. Making a proper handling of duplicated activities impossible, outside of decreasing precision through overgeneralization. This argument is supported by a 50% increase of False Positives when duplicated activities are enabled compared to being disabled, indicating underfitting models. Similarly, the drop in long-term dependencies can be attributed to how the IM splits the log. This leads to the dependencies between activities being lost once they are split into different partitions.

5.3 Real-life datasets

In addition to evaluating OptIMIIst on synthetic benchmark data, its performance on real-life event data is equally important, to determine its effectiveness in practical applications. While synthetic datasets provide controlled environments to evaluate algorithms under predefined conditions, they may not fully capture the complexity and variability inherent in real-world processes. Real-life event logs often contain noise, missing data, and natural irregular patterns that synthetic logs may not simulate effectively. Moreover, synthetic datasets are typically designed to highlight specific features of process discovery algorithms, which might not reflect the broader range of challenges encountered in real-world scenarios. Therefore, evaluating Op-

tIMIIst on real-life data is essential to ensure that its generated process models achieve high fitness and precision when applied to practical use cases.

To achieve this, the publicly available BPIC datasets serve as a valuable resource, providing diverse, real-world event logs. The event logs from the BPI Challenges of 2012[8], 2013[9], 2017[10], and 2020[11], as well as the Sepsis log[12] have been selected.

5.3.1 Setup

Besides OptIMIIst, this part of the evaluation is also run on the IMf as a baseline algorithm, the AIM as one of the similar state-of-the-art IM variants, and the ILP-Miner to include one discovery algorithm outside the realm of IM. For the IMf and ILP-Miner, the implementations from the pm4py library have been used. For AIM, a propitiatory implementation kindly provided by Celonis S.E., for the sole purpose of this thesis, has been used.

To construct a robust evaluation framework, following some proposals from [27], the original log is randomly split into a train/test split with 80% of cases in the training set and 20% of cases in the testing set. Models were discovered on the training set, and fitness was evaluated using the test set using alignment-based fitness. Additionally, alignment-based precision of the models was measured using the entire log. The size of the mined Petri nets is measured by counting the amount of places, transitions, and arcs. This means the IM based models must be translated to Petri nets using the tooling pm4py provides. Additionally, activity completeness is measured by counting the number of activities in the log compared to the activities present in the models. To evaluate if OptIMIIsts fallthroughs have an effect on mining, or the decisions reside in the IM framework, the amount of times the fallthrough is triggered is counted. Similarly, the ILP-Miner is the only miner that does not produce guaranteed sound models. For these, a soundness check is added. Finally, the pure runtime spent on the discovery of each miner is measured. This runtime does not include the translation of the output model to Petri nets. The experiment is repeated five times with random train/test splits to mitigate the effect of randomness in the split, and the results are averaged.

5.3.2 Results

Table 5.1 displays the aggregated evaluation results. The average performances of the miners normalized and averaged over all logs can also be found on a radar chart in Figure 5.2. Not displayed is the number of fallthrough occurrences of OptIMIIst; it is used at least once for each log and more than five times for all logs except the $BPIC_{2013}$ and Sepsis logs. The radar chart supports the argument that OptIMIIst balances different quality dimensions particularly well and achieves competitive results compared to the other miners.

[8] $BPIC_{12}$ http://doi.org/10.4121/UUID:3926DB30-F712-4394-AEBC-75976070E91F
[9] $BPIC_{13}$ http://doi.org/10.4121/UUID:A7CE5C55-03A7-4583-B855-98B86E1A2B07
[10] $BPIC_{17}$ http://doi.org/10.4121/UUID:5F3067DF-F10B-45DA-B98B-86AE4C7A310B
[11] $BPIC_{20}$ http://doi.org/10.4121/UUID:52FB97D4-4588-43C9-9D04-3604D4613B51
[12] Sepsis http://doi.org/10.4121/UUID:915D2BFB-7E84-49AD-A286-DC35F063A460

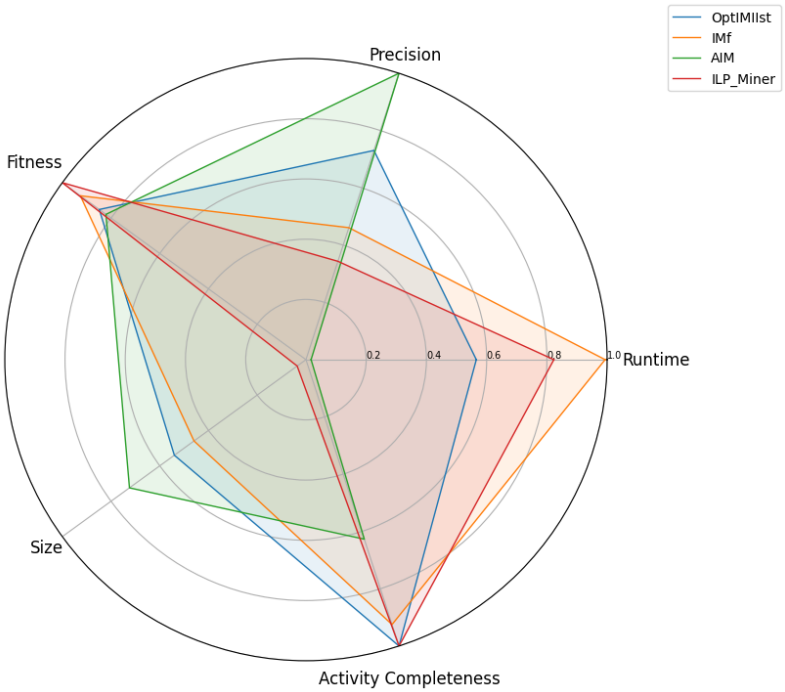


Figure 5.2: Radar Chart of the performance of the miners on real life event logs

It can be seen that OptIMIIst performs similarly well in terms of fitness as the IMf with worse runtime, but a significant increase in Precision. The slightly worse average fitness visible in the Radar chart can be attributed to the performance on two specific logs, BPIC_{2012} and BPIC_{2017} on which OptIMIIst has significantly worse fitness but significantly better precision. When calculating the F_1 score of IMf on BPIC_{2012}, the result is only 0.28 while OptIMIIst scores 0.75, a significant improvement. When looking at the simplicity of the nets OptIMIIst produces in comparison to IMf, like the examples in Figure A.1a and A.1b, the nets of OptIMIIst are usually not only smaller but because not making use of any global fallthroughs like the *activity once pre trace* also more globally sequential. As such, the author argues that OptIMIIst effectively outperforms the IMf, specifically regarding adaptability to different challenges observed in real-life event logs.

When comparing OptIMIIst with the ILP-Miner, the first observation is that the ILP-Miner always archives a perfect or near-perfect fitness, outperforming in that regard but also producing very large underfitting models with poor precision. It should be noted that when applying the ILP-Miner, it is usually advisable to use filtering beforehand. Doing so would add some balance between the fitness and precision measures but also require experimentation to find a suitable filter threshold. The experimentation is something not required for OptIMIIst, which dynamically balances fitness and precision without requiring any input parameters from the user to archive competitive results. The ILP-Miner is the only algorithm evaluated not guaranteeing soundness, with only 30% of mined models being sound.

Finally, the AIM was used to compare OptIMIIsts performance with another advanced version of the IM. When only looking at the classic performance criteria of fitness, size, and precision AIM outperforms OptIMIIst on almost all of the logs, with

only a few exceptions. However, when looking at how many activities the output models of AIM contain, the effect of its activity filtering is visible. AIM applies activity filtering at each call of the AIM fallthrough and can in some cases remove more than 50% of the activities from the original log. This leads to an intended trade-off between precision, simplicity and activity completeness. For example, in BPIC 2020_{DD}, the models mined by AIM only contain 7 of the original 17 activities. This can also be seen when comparing Figure A.2a and Figure A.2b. AIMs model only encapsulates a single sequential order of activities, with two being optional through a τ-skip. This issue is not present in OptIMIIst. OptIMIIst has no activity filtering capabilities and, as such, never drops activities and still upholds similar fitness and competitive precision. OptIMIIst's model shows this by enabling more behaviour with clearly showing different stages of the process enabling options in its behaviour.

Surprisingly, the runtime of AIM is, with one exception, worse than the one of OptIMIIst by a factor of at least two in most cases. While OptIMIIst solves potentially computationally more expensive ILP problems for cut detection than AIMs clustering approach, this downside does not translate to real-life problem instances.

Table 5.1: OptIMIIst, AIM, IMf, ILP-Miner Benchmarks

Event Log	Algorithm	Runtime (s)	Precision	Fitness	Size	Activities
BPIC 2012	OptIMIIst	80.20	0.902	0.649	198	24.0
	IMf	**5.60**	0.165	0.998	239	23.8
	AIM	244.65	**0.927**	0.592	162	21.0
	ILP-Miner	20.29	0.132	**1.000**	291	24.0
BPIC 2013_{CP}	OptIMIIst	0.86	0.927	0.958	36	4.0
	IMf	**0.01**	0.970	0.989	50	3.8
	AIM	10.92	**0.999**	0.771	16	3.0
	ILP-Miner	0.08	0.883	**1.000**	43	4.0
BPIC 2013_{I}	OptIMIIst	10.30	0.982	0.905	48	4.0
	IMf	**0.07**	0.843	0.964	49	4.0
	AIM	30.21	**0.995**	0.905	37	3.0
	ILP-Miner	0.87	0.684	**1.000**	39	4.0
BPIC 2017	OptIMIIst	769.24	**0.965**	0.742	186	26.0
	IMf	**20.92**	0.375	0.949	285	25.2
	AIM	655.36	**0.965**	0.643	116	21.2
	ILP-Miner	230.33	0.172	**1.000**	632	26.0
BPIC 2020_{DD}	OptIMIIst	15.55	0.600	0.942	122	16.8
	IMf	**0.03**	0.258	0.936	125	15.6
	AIM	38.96	**1.000**	0.910	36	7.0
	ILP-Miner	0.56	0.157	**1.000**	252	16.8
BPIC 2020_{ID}	OptIMIIst	32.41	0.378	0.878	269	33.6
	IMf	**0.09**	0.202	0.886	284	29.0
	AIM	66.22	**0.951**	0.882	89	14.2
	ILP-Miner	6.19	0.136	**1.000**	1213	33.6
BPIC 2020_{PTC}	OptIMIIst	6.53	0.294	0.872	236	29.0
	IMf	**0.04**	0.243	0.879	243	26.4
	AIM	23.55	**0.562**	0.889	107	13.0
	ILP-Miner	3.00	0.237	**0.999**	913	29.0
BPIC 2020_{RFP}	OptIMIIst	10.78	0.371	0.925	146	18.6
	IMf	**0.02**	0.272	0.913	157	16.4
	AIM	25.94	**0.851**	0.925	69	10.4
	ILP-Miner	0.57	0.194	**1.000**	324	18.6
BPIC 2020_{TPD}	OptIMIIst	66.22	0.271	0.800	453	50.4
	IMf	**0.45**	0.199	0.774	410	41.2
	AIM	84.36	**0.727**	0.800	137	18.0
	ILP-Miner	86.31	0.081	**1.000**	3019	50.4
Sepsis	OptIMIIst	3.53	0.727	0.846	113	16.0
	IMf	**0.05**	0.602	0.970	152	15.0
	AIM	16.39	**0.828**	0.890	105	14.0
	ILP-Miner	1.52	0.398	**1.000**	296	16.0

Chapter 6

Discussion

This chapter reflects on the findings presented in the evaluation and positioning the contributions of this thesis within the field of process mining. It discusses the implications of the proposed approaches for process mining research and practice while critically examining their limitations. Additionally, potential future directions are outlined to guide subsequent work in this area.

Key Contributions:

OptIMIIst makes two main contributions to the field of process discovery, besides the methods as a whole. The first is integrating optimization problems into the IM framework for cut detection. Unlike AIM and PIM, which rely on heuristics to identify cut candidates, OptIMIIst uses ILPs to find optimal versions of specific cuts. This approach demonstrates greater adaptability, allowing customization through modifications to the objective function or by adding constraints. Moreover, the flexibility of ILPs can potentially address issues such as duplicate activities or long-term dependencies with tailored formulations.

The second contribution is estimating fitness and precision at a local level for incomplete process trees. To the author's knowledge, no prior work offers such measures. These estimators were shown to be effective for scoring cut candidates in the evaluation of the proposed miner. But further evaluation is required to assess their applicability to broader problems beyond cut scoring.

OptIMIIst does not require any input parameters out of the event log and has the ability to handle infrequent and incomplete behaviour without relying on explicit monotonic filtering.

Limitations

Despite its strengths, OptIMIIst has some limitations. For one, the ILPs introduced for cut detection only guarantee locally optimal solutions. They do not guarantee that these decisions lead to the globally best process tree.

This issue also extends to the cut scoring, an issue not unique to OptIMIIst but also affects AIM and PIM. Ideally, all cut candidates could be further branched, and conformance checking could be applied to identify the subtree that best represents

the event log. However, this would result in an exponential increase in computational complexity, making the approach impractical.

To address this complexity, OptIMIIst relies on local scoring of cuts. While this strategy improves feasibility, no guarantees for global optimality can be achieved without a backtracking mechanism. Future enhancements could involve dynamic evaluation of subtrees to assess their viability iteratively. This approach could leverage this thesis's fitness and precision estimators to compare subtrees and prioritize promising candidates.

Additionally, in contrast to AIM, OptIMIIst cannot handle infrequent behaviour on an activity level. Issues like wrong labels on activities can not be detected and, as such, not handled at all by OptIMIIst.

Also the potential high complexity of solving ILPs is a concern for applicability of the method in practice. While the performance on real-life event logs during the evaluation was competitive, the high complexity is still an existing concern.

Evaluation Results

The evaluation in Chapter 5 demonstrated that OptIMIIst performs comparably to state-of-the-art process discovery algorithms, such as AIM and PIM, on real-life event logs and synthetic benchmarks. On the PDC datasets, OptIMIIst showed adaptability to many different log structures and effectively handled noise in the data. However, like AIM, OptIMIIst has issues to address long-term dependencies and duplicate activities. While AIM sacrifices activity completeness for improved fitness and precision, OptIMIIst achieves comparable fitness and precision without this additional drawback. OptIMIIst also appears to handle loops more effectively than AIM.

Interestingly, despite the computational cost of solving ILPs, OptIMIIst exhibited faster runtimes than AIM on real-life event logs. This performance, achieved with an unoptimized implementation, highlights the potential for further runtime improvements. Future implementations could explore alternative ILP solvers, such as the open-source SCIP or other frameworks like Satisfiability modulo theories (SMT) solvers. While SMT solvers generally focus on satisfiability rather than optimization, some, like Z3, offer optimization extensions that could be explored.

Future Directions

Several promising directions for future research emerge from this thesis. One critical area involves improving the handling of long-term dependencies and duplicate activities, which continue to present challenges for process discovery algorithms. New ILP formulations or heuristic adjustments could provide more robust solutions for these issues. This would require additional research into how these ILPs could be constructed and changes to the log splitting. Log splitting would be required to accommodate activity instances assigned to different partitions without the cut detection supporting them. Long-term dependencies, on the other hand, are more complex to approach. A reason for techniques from the family of IM to struggle with them is the lack of techniques to model dependencies between distant subtrees in process trees. An extension to process trees could be envisioned with added

dependency relations, making the discovery of long-term dependency possible.

Like duplicate activities, activity filtering could also be integrated into the ILPs. While the information loss induced by activity filtering by AIM on the real-life event data was quite severe, it is still an effective way to address infrequent behaviour on activity level. Integrating activity filtering into the ILPs could offer an avenue to perform targeted filtering of the most problematic activities unrelated to their frequency. The approach promises to reduce the information loss on rare but unproblematic activities.

Furthermore, dynamic subtree evaluation represents an intriguing possibility. By iteratively comparing multiple promising cuts, it might be possible to identify and prioritize the most viable subtrees. Creating a middle path between evaluating all cut candidates and continuing only with the candidate with the highest fitness and precision estimation.

Also, the used F_1 score to create the aggregate fitness and precision estimation could offer configurability. While Optimist's goal is to balance fitness and precision, specific use cases could be interested in a more fitness or precision-focused model. The cut scoring as well as adaptations to the ILPs are an area where such preferences could be handled.

Exploring alternative optimization frameworks is another potential direction. While ILPs have shown significant promise, frameworks such as SMT solvers offer an interesting comparative avenue. Although SMT solvers typically focus on satisfiability, some, like Z3, have optimization extensions that could be investigated. Comparing their performance with ILPs might uncover new efficiencies or capabilities. Additionally, further improvements to runtime could be achieved by optimizing the implementation of OptIMIIst and experimenting with different solvers, such as the open-source SCIP. Specifically in the ILP, the impact of the symmetry-breaking constraints on solver performance can be explored.

Another avenue, without the goal of improving OptIMIIst, involves conducting an in-depth evaluation of the fitness and precision estimators themselves. While these measures proved useful for cut scoring, their broader applicability to other problems within process discovery remains unexplored. Extending their validation could enhance their utility across various contexts.

Chapter 7

Related Works

In Section 2.2 the concept of process discovery has been introduced, with the IM as an concrete example in Section 3.5 of the previous section. Over the years different process discovery algorithms have been developed. Some with a focus on specific issues and some attempting to solve the general problem of process discovery. Early examples include the α-Miner [1] and its extension the α+-Miner [13], which adds support for short loops. These miners discover workflow models but are generally considered unsuitable for larger processes, and have since been overshadowed by more advanced discovery techniques. The heuristic miner [34] offers adaptability by taking the frequencies of the directly follows relations into account. However the miner in its original version does not carry any soundness guarantees. Similar is true for other discovery algorithms in the area like the est-Miner [24] or the ILP-Miner [35]. The later discovers a Petri net with a minimum of places using ILP, grounded in the theory of language regions similar to the approach in [8]. While extensions of the ILP-Miner [36] offer relaxed soundness by adapting the mining approach, giving some behavioural guarantees, they fall short of proper soundness. Extensions of the heuristic miner include Fodina [32] and the Structured Miner [6]. The former, removes some types of deadlocks the heuristic miner is likely to introduce. The latter uses the heuristic miner to discover an initial model and applies a preprocessing to arrive at better output models, that are often sound. This approach is based on the hypothesis that discovering an expressive potentially unstructured and unsound model first and only afterwards transforming it into a structured, potentially sound one results in better models.

On the contrary the family of IM algorithms and the Split Miner [5] guarantee soundness of the discovered model by trading a representational bias for this guarantee. This is an guarantee OptIMIIst, as an algorithm in the family of IM algorithms, carries in contrast to the previously mentioned algorithms. As such versions and extensions of the IM discover process trees, which, as described in section 3.4, guarantee to be transformed into sound WF-nets. This by extension includes other Miners that discover process trees. The Evolutionary Tree Miner (ETM) [11] employs an genetic approach to discovering process trees. Local process models [30] are a bottom up approach to process discovery, while the Indulpet Miner [22] combines concepts from IM, ETM and local process models. As a future direction of research, a combination of OptIMIIst with the bottom up approach of local process

model in the Indulpet Miner might show promise.

Besides these more ambitious applications of process trees, several versions and extensions of the IM have been developed. IMf, instead of applying a fallthrough, applies filtering of the most infrequent directly follows relations up to a preset threshold. This enables some flexibility to handle noise in the data. However if the IMf still cannot identify a valid cut after filtering, it falls back to fallthroughs like the original IM [20]. While the IMf generally performs better in terms of precision by sacrificing some fitness, it struggles with incomplete or very noisy data. Additionally being reliant on the flower model, as a potential fallthrough, IMf might sacrifice precision early in the recursion [20]. The IMc [21] algorithm on the other hand enhances the IM framework by addressing incomplete behaviour. When no definition-fulfilling cut is available, it employs probabilistic cut quality estimates to identify suitable cuts. However, the algorithm faces challenges in real-life applications due to its large search space for potential cuts. Additionally, it as well defaults to a flower model when no suitable cut is found, limiting its practical effectiveness. OptIMIIst differs in its approach from both, not only targetting either only infrequency or only incompleteness, but handling both at the same time.

The differentially private Inductive Miner [10] is a version of the IM that applies abstractions to the log to guarantee an ϵ-differentially private property. While required for specific use cases this is not a property OptIMIIst supports.

Handling both infrequent and incomplete behaviour is a property that OptIMIIst shares with the most recent advancements on the IM framework like PIM [9] and AIM [14]. Similar to the IMc both of which also use probabilistic cut quality estimates to find cuts if the IM framework does not find one. One major difference is that these techniques replace the fallthrough functionality with a single technique always returning a cut, not making use of the flower model at all. As such PIM combines concepts from IMf and IMc. Similar to IMf, it applies percentile-based filtering to preprocess the log. It then calculates activity scores for all activity pairs using measures inspired by IMc [9]. Heuristics are used to prune the search space of suitable cut candidates, which is naively searched for the best cut to apply.

AIM applies a similar approach, calculating activity scores unique for each operator between activities. However instead of applying a unguided search for the best cut like PIM or IMc, AIM employs a clustering mechanism, to find the best cut for each operator. Similar to OptIMIIst, AIM calculates scores of these candidates that are compared to find the best cut candidate. Additionally AIM employs activity filtering on each level of the recursion in which IM cannot find a cut. This is a significant change from the filtering of PIM and IMf which only filter the DFG on arc level. But for both the filtering remains "monotonic". Both algorithms apply filtering based on the most infrequent behaviour instead of specifically filtering or disregarding problematic behaviour. For example AIM performs its cut detection multiple times on different activity filtering levels, again selecting the best one in the end. Its step size remains the only input parameter AIM requires, besides the event log.

Both AIM and PIM demonstrate strong performance in practice. Their main strength results from balancing fitness and precision better than alternative discovery algorithms and without requiring a large set of input parameters or experimenta-

tion. Both employ a similar approach to OptIMIIst, using cut scores and employing heuristics to prune and search the solution space for the best cut. Both also rely on monotonic filtering and the quality of the applied heuristics.

Chapter 8

Conclusion

In this thesis a novel process discovery algorithm called OptIMIIst has been introduced. OptIMIIst is a method which balances the different quality dimensions while effectively handling infrequent and incomplete behaviour. The method consists of a twofold approach for identifying locally optimal process trees for cases in which the IM framework is unable to find a base case or cut. The method archives this by finding locally optimal activity partitions for each operator supported in the IM framework, relaxing the formal definitions to take incomplete and infrequent behaviour into account while detecting the cuts. A customized fitness and precision estimation enables the algorithm to find the most suitable cut from the candidate set. An enhanced handling of empty traces on base case level wraps up the approach.

OptIMIIst was implemented in python, using Gurobi as an optimization backend to solve the ILPs used in the cut discovery stage. When evaluating OptIMIIst on the Process Discovery Contest 2024 data it showed adaptability to a wide range of log and model challenges including noise and loop detection. Challenges remain the correct detection of duplicated activities and long term dependencies, an area in which IM based algorithms typically struggle with. Additionally OptIMIIst was evaluated on real-life event data demonstrating its competitive performance in terms of precision, fitness and simplicity compared to the existing algorithms IMf, IMf and ILP-Miner. Notably OptIMIIst did not require or used any additional filtering, either inbuilt into the algorithm or applied as preprocessing.

As future directions for research a better handling of duplicate activities and long-term dependencies was identified. Duplicate activities could theoretically be integrated into the ILP cut finding, enabling versions of the ILPs to assign one activity to multiple partitions. Long-term dependencies on the other hands would require larger changes to the formalism of process trees.

Besides tackling these structural challenges a measured approach to activity filtering might also be a path to further improve the performance of OptIMIIst. AIM, which outperforms OptIMIIst in most real life use cases, was argued to add a too severe amount of activity filtering, indicating that the information loss might not be handled sufficiently. Integrating activity filtering into the ILPs, identifying specific problematic activities, might be a way to mitigate this issue. But also optimizations to the implementation of the optimization problems, by using different solvers like SMT-solvers or open-source tools and evaluating the effect of symmetry breaking

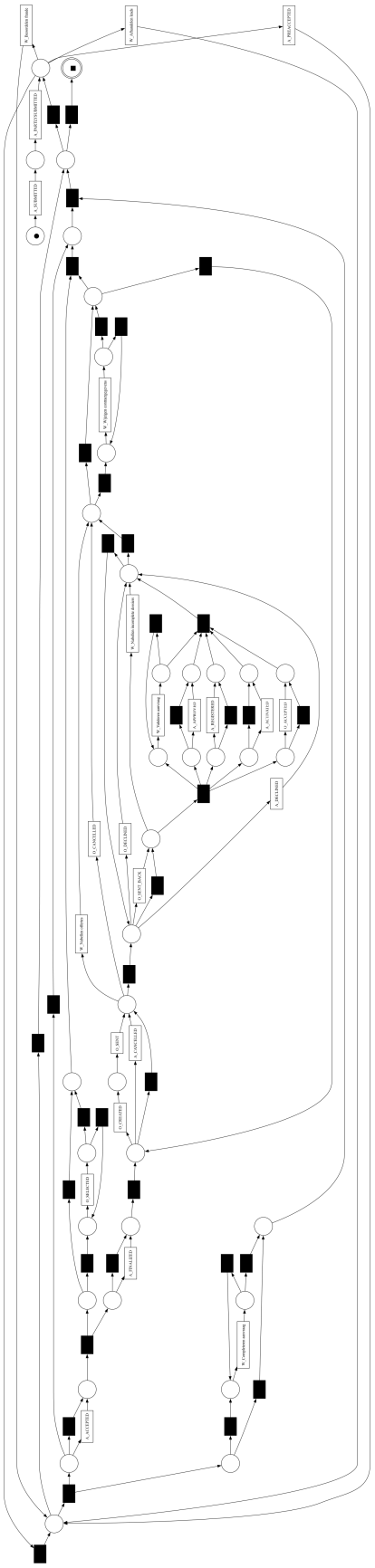
constraint on solving times is a possible path for future work.

In conclusion, the OptIMIIst framework, along with its techniques for cut detection and evaluation, demonstrates competitive performance against other state-of-the-art process discovery algorithms. It promises to handle incomplete and infrequent behaviour without the need for user input or filtering and opens avenues for further research. The ILP-based approach to cut detection provides a promising foundation that aligns closely with the cut definitions of IM. It offers straightforward possibilities for extensions and adaptations to accommodate different process mining requirements.

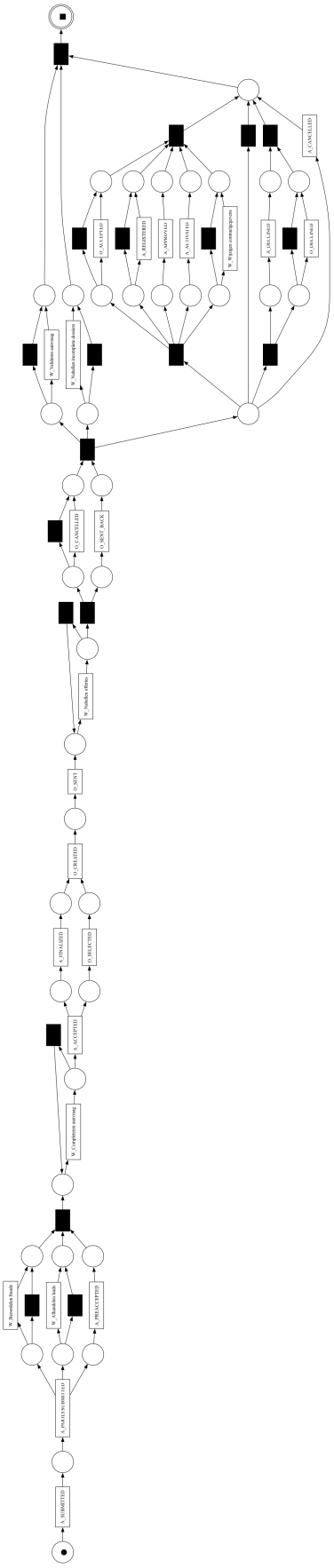
Appendix A

Petri nets from evaluation

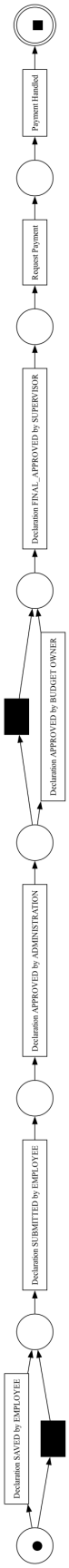
To give the displayed process models as much space as possible, they start on the following page.



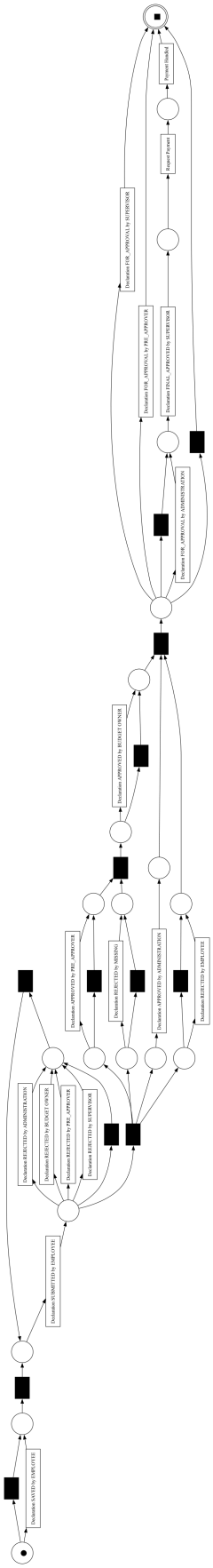
(a) IMf Petri Net from $BPIC_{2012}$



(b) OptIMIIst Petri Net from $BPIC_{2012}$



(a) AIM Petri Net from BPIC$_{2020DD}$



(b) OptIMIIst Petri Net from BPIC$_{2020DD}$

Bibliography

[1] van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering **16**(9), 1128–1142 (2004). https://doi.org/10.1109/TKDE.2004.47

[2] van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Form. Asp. Comput. **23**(3), 333–363 (May 2011). https://doi.org/10.1007/s00165-010-0161-4, `https://doi.org/10.1007/s00165-010-0161-4`

[3] van der Aalst, W.M.P.: Process mining: A 360 degree overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook, Lecture Notes in Business Information Processing, vol. 448, pp. 3–34. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_1, `https://doi.org/10.1007/978-3-031-08848-3_1`

[4] van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining Knowl. Discov. **2**(2), 182–192 (2012). https://doi.org/10.1002/WIDM.1045, `https://doi.org/10.1002/widm.1045`

[5] Augusto, A., Conforti, R., Dumas, M., La Rosa, M.: Split miner: Discovering accurate and simple business process models from event logs. In: 2017 IEEE International Conference on Data Mining (ICDM). pp. 1–10 (2017). https://doi.org/10.1109/ICDM.2017.9

[6] Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Bruno, G.: Automated discovery of structured process models from event logs: The discover-and-structure approach. Data & Knowledge Engineering **117**, 373–392 (2018). https://doi.org/https://doi.org/10.1016/j.datak.2018.04.007, `https://www.sciencedirect.com/science/article/pii/S0169023X18301708`

[7] Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4) (2019)

[8] Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) Business Process Management, 5th International Conference, BPM 2007, Brisbane,

Australia, September 24-28, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4714, pp. 375–383. Springer (2007). https://doi.org/10.1007/978-3-540-75183-0_27, https://doi.org/10.1007/978-3-540-75183-0_27

[9] Brons, D., Scheepens, R., Fahland, D.: Striking a new balance in accuracy and simplicity with the probabilistic inductive miner. In: Ciccio, C.D., Francescomarino, C.D., Soffer, P. (eds.) 3rd International Conference on Process Mining, ICPM 2021, Eindhoven, The Netherlands, October 31 - Nov. 4, 2021. pp. 32–39. IEEE (2021). https://doi.org/10.1109/ICPM53251.2021.9576864, https://doi.org/10.1109/ICPM53251.2021.9576864

[10] Brons, D., Scheepens, R., Fahland, D.: Striking a new balance in accuracy and simplicity with the probabilistic inductive miner. In: Ciccio, C.D., Francescomarino, C.D., Soffer, P. (eds.) 3rd International Conference on Process Mining, ICPM 2021, Eindhoven, The Netherlands, October 31 - Nov. 4, 2021. pp. 32–39. IEEE (2021). https://doi.org/10.1109/ICPM53251.2021.9576864, https://doi.org/10.1109/ICPM53251.2021.9576864

[11] Buijs, J., van Dongen, B., van der Aalst, W.: A genetic algorithm for discovering process trees. In: 2012 IEEE Congress on Evolutionary Computation. pp. 1–8 (2012). https://doi.org/10.1109/CEC.2012.6256458

[12] Dantzig, G.B., Thapa, M.N.: The Simplex Method, pp. 63–111. Springer New York, New York, NY (1997). https://doi.org/10.1007/0-387-22633-8_3, https://doi.org/10.1007/0-387-22633-8_3

[13] De Medeiros, A.A., Van Dongen, B.F., Van der Aalst, W.M., Weijters, A.J.M.: Process mining: extending the alpha-algorithm to mine short loops. Technische Universiteit Eindhoven, Eindhoven University of Technology, Eindhoven (2004)

[14] van Detten, J.N., Schumacher, P., Leemans, S.J.J.: An approximate inductive miner. In: 5th International Conference on Process Mining, ICPM 2023, Rome, Italy, October 23-27, 2023. pp. 129–136. IEEE (2023). https://doi.org/10.1109/ICPM60904.2023.10271971, https://doi.org/10.1109/ICPM60904.2023.10271971

[15] Dikin, I.I.: Iterative solution of problems of linear and quadratic programming. Sov. Math., Dokl. **8**, 674–675 (1967)

[16] Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018). https://doi.org/10.1007/978-3-662-56509-4, https://doi.org/10.1007/978-3-662-56509-4

[17] Karunaratne, A., Polyvyanyy, A., Moffat, A.: The role of log representativeness in estimating generalization in process mining. In: 6th International Conference on Process Mining, ICPM 2024, Kgs. Lyngby, Denmark, October 14-18, 2024. pp. 33–40. IEEE (2024). https://doi.org/10.1109/ICPM63005.2024.10680679, https://doi.org/10.1109/ICPM63005.2024.10680679

[18] Leemans, S.J.J.: Robust Process Mining with Guarantees - Process Discovery, Conformance Checking and Enhancement, Lecture Notes in Business Information Processing, vol. 440. Springer (2022). https://doi.org/10.1007/978-3-030-96655-3, https://doi.org/10.1007/978-3-030-96655-3

[19] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - A constructive approach. In: Colom, J.M., Desel, J. (eds.) Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7927, pp. 311–329. Springer (2013). https://doi.org/10.1007/978-3-642-38697-8_17, https://doi.org/10.1007/978-3-642-38697-8_17

[20] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers. Lecture Notes in Business Information Processing, vol. 171, pp. 66–78. Springer (2013). https://doi.org/10.1007/978-3-319-06257-0_6, https://doi.org/10.1007/978-3-319-06257-0_6

[21] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: Ciardo, G., Kindler, E. (eds.) Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8489, pp. 91–110. Springer (2014). https://doi.org/10.1007/978-3-319-07734-5_6, https://doi.org/10.1007/978-3-319-07734-5_6

[22] Leemans, S.J.J., Tax, N., ter Hofstede, A.H.M.: Indulpet miner: Combining discovery algorithms. In: Panetto, H., Debruyne, C., Proper, H.A., Ardagna, C.A., Roman, D., Meersman, R. (eds.) On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11229, pp. 97–115. Springer (2018). https://doi.org/10.1007/978-3-030-02610-3_6, https://doi.org/10.1007/978-3-030-02610-3_6

[23] Little, J.D., Murty, K.G., Sweeney, D.W., Karel, C.: An algorithm for the traveling salesman problem. Operations research **11**(6), 972–989 (1963)

[24] Mannel, L.L., van der Aalst, W.M.P.: Finding complex process-structures by exploiting the token-game. In: Donatelli, S., Haar, S. (eds.) Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11522, pp. 258–278. Springer (2019). https://doi.org/10.1007/978-3-030-21571-2_15, https://doi.org/10.1007/978-3-030-21571-2_15

[25] Margot, F.: Symmetry in integer linear programming. In: Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.) 50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art, pp. 647–686. Springer (2010). https://doi.org/10.1007/978-3-540-68279-0_17, https://doi.org/10.1007/978-3-540-68279-0_17

[26] Petri, C.: Kommunikation mit Automaten. Rheinisch-Westfälisches Institut für Instrumentelle Mathematik Bonn: [Schriften des Rheinisch-Westfälischen Instituts für Instrumentelle Mathematik, Rheinisch-Westfälisches Institut f. instrumentelle Mathematik an d. Univ. (1962), https://books.google.de/books?id=NCZMvAEACAAJ

[27] Rehse, J.R., Leemans, S., Fettke, P., Van der Werf, J.M.: On process discovery experimentation. ACM Transactions on Software Engineering and Methodology (11 2024). https://doi.org/10.1145/3672447

[28] Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1), 64–95 (2008). https://doi.org/https://doi.org/10.1016/j.is.2007.07.001, https://www.sciencedirect.com/science/article/pii/S030643790700049X

[29] Suriadi, S., Andrews, R., ter Hofstede, A., Wynn, M.: Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. Information Systems **64**, 132–150 (2017). https://doi.org/https://doi.org/10.1016/j.is.2016.07.011, https://www.sciencedirect.com/science/article/pii/S0306437915301344

[30] Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.: Mining local process models. Journal of Innovation in Digital Ecosystems **3**(2), 183–196 (2016). https://doi.org/https://doi.org/10.1016/j.jides.2016.11.001, https://www.sciencedirect.com/science/article/pii/S2352664516300232

[31] van der Aalst, W.M.: A practitioner's guide to process mining: Limitations of the directly-follows graph. Procedia Computer Science **164**, 321–328 (2019). https://doi.org/https://doi.org/10.1016/j.procs.2019.12.189, https://www.sciencedirect.com/science/article/pii/S1877050919322367, cENTERIS 2019 - International Conference on ENTERprise Information Systems / ProjMAN 2019 - International Conference on Project MANagement / HCist 2019 - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN/HCist 2019

[32] vanden Broucke, S.K., De Weerdt, J.: Fodina: A robust and flexible heuristic process discovery technique. Decision Support Systems **100** (2017). https://doi.org/https://doi.org/10.1016/j.dss.2017.04.005, https://www.sciencedirect.com/science/article/pii/S0167923617300647, smart Business Process Management

[33] Weerdt, J.D., Wynn, M.T.: Foundations of process event data. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook, Lecture Notes in Business Information Processing, vol. 448, pp. 193–211. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_6, https://doi.org/10.1007/978-3-031-08848-3_6

[34] Weijters, A.J.M.M., van der Aalst, W.M.P., Alves de Medeiros, A.K.: Process Mining with the Heuristics Miner-algorithm. Technische Universiteit Eindhoven, Eindhoven University of Technology, Eindhoven (2006), https://publications.rwth-aachen.de/record/714920

[35] van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. Fundam. Informaticae **94**(3-4), 387–412 (2009). https://doi.org/10.3233/FI-2009-136, https://doi.org/10.3233/FI-2009-136

[36] van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering workflow nets using integer linear programming. Computing **100**(5), 529–556 (2018). https://doi.org/10.1007/S00607-017-0582-5, https://doi.org/10.1007/s00607-017-0582-5