Locally Optimized Process Tree Discovery

Calvin Schröder[1]✉, Jan Niklas van Detten[1,3], and
Sander J.J. Leemans[1,2][0000−0002−5201−7125]

[1] RWTH Aachen University, Aachen, Germany
[2] Fraunhofer FIT, Germany
[3] Celonis, Munich, Germany

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, we propose a new process discovery technique called OptIMIIst, which 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.

Keywords: Process Mining · Process Discovery · Process Trees.

1 Introduction

In business processes activities and resources are coordinated to achieve organizational goals. Understanding and optimizing these processes is one way to enhance efficiency. Traditionally, process analysis has relied on manual construction of models in standardized formalisms, like Petri nets. However, with the advent of digital technologies, processes generate detailed automated records of their execution, called event logs. Process discovery, a central field of process mining, aims to automatically derive process models from these event logs.

For a process model to effectively aid the analysis of the underlying process, it must meet certain quality criteria, evaluated using four dimensions. Fitness measures how well the model captures the actual behaviour recorded in the event log, while precision ensures the model accurately represents only the observed behaviour, avoiding unnecessary generalisation. In addition to these quantifiable metrics, soundness ensures that the model is behaviourally correct [15,2,18], while simplicity, though not directly measurable, keeps the model straightforward and easy to understand.

Key challenges in process discovery include infrequent and incomplete behaviour, both of which affect the quality of models [16]. Infrequent behaviour refers to behaviour rarely observed in the log, such as noise resulting from data entry errors or rarely happening exceptions. On the one hand, process discovery techniques must avoid noisy elements to avoid overfitting and focus on capturing the "core" process. On the other hand, discovery techniques must be robust against incompleteness as we cannot assume to have recorded all behaviour that is possible in the process in the event log.

The Inductive Miner (IM) [10] is a process discovery algorithm that recursively decomposes an event log into smaller logs to construct a process tree, which can be transformed into sound workflow nets. However, balancing the remaining model qualities fitness, precision, and simplicity while handling incomplete and infrequent behaviour is particularly challenging. If no cut can be found, IM needs to resort to fallthroughs, which sacrifice precision for fitness. Existing approaches such as the Probabilistic Inductive Miner (PIM) [5] and the Approximate Inductive Miner (AIM) [7] attempt to address these issues by filtering out infrequent behaviour before applying heuristics, which deviate from formal cut definitions. While the Inductive Miner Incomplete (IMc) [12] only handles incomplete behaviour. This highlights a gap for a process discovery technique that manages both infrequent and incomplete behaviour simultaneously, while adhering closely to the formal cut definitions.

In this paper, we propose a new process discovery technique that guarantees soundness and addresses both infrequent and incomplete behaviour, by generalising the cut definitions of the IM framework. First, we choose one candidate cut for each process tree operator that is locally optimal with respect to the generalised cut definition. Second, we use an estimation of local fitness and precision to choose the best of the candidate cuts. We call our new discovery technique the Opt-IM-II-st (Optimization-InductiveMiner-Infrequent-Incomplete-eSTimation). We evaluate the model quality of OptIMIIst with respect to other state-of-the-art algorithms using using real-world event logs, which shows that OptIMIIst has a competitive performance.

The paper is structured as follows: Section 2 introduces relevant related work, Section 3 introduces the basics of process trees and IM. Section 4 details OptIMIIst and its implementation. Section 5 benchmarks and evaluates OptIMIIst. Finally, Section 6 summarizes our findings and presents final thoughts.

2 Related Work

The Inductive Miner employs a recursive approach to process discovery, limiting itself to mining process trees. Although the IM introduces a representational bias, it offer the advantage of ensuring soundness when converted to Petri nets [10]. Several discovery algorithms implemented the IM, to address specific challenges. A common way to handle infrequent behaviour is employing filtering, for example in the Inductive Miner infrequent (IMf) [11]. Through the optimization, OptIMIIst, specifically targets and “filters” only the violating behaviour,

thereby minimizing information loss in comparison to frequency-based filtering. IMc focuses on incomplete event logs, but its application can be infeasible [12]. The Evolutionary Tree Miner (ETM) [6] employs a genetic algorithm and evaluates the entire process trees based on fitness and precision, guiding the evolution of the population accordingly, but as an optimization meta-heuristics does not guarantee optimality. In contrast, OptIMIIst estimates fitness and precision at a local level, focusing on optimizing individual partitions rather than the entire tree. The Indulpet Miner [13] is a combination of four process discovery techniques including ETM and IM. The most closely related versions to the OptIMIIst are AIM [7] and PIM [5], both of which handle infrequent and incomplete behaviour through quality measures. While PIM searches all possible cuts for the best cut, AIM uses a clustering approach. For its filtering OptIMIIst implicitly filters during optimization while PIM employs a similar filtering as IMf. AIM handles filtering at the activity level through parameter optimization. Besides OptIMIIst other techniques also use optimization as a tool for process discovery but do not guarantee soundness [19,4,17].

In literature, a manifold of other techniques for process discovery have been introduced. For a review of earlier advancements in the area of process discovery and a comprehensive comparison, we refer to [3]. Newer approaches include the eST-Miner mining Petri nets by searching for fitting places [14].

3 Preliminaries

An event log L is a record of traces σ, where each trace $\sigma = \langle a_1, \ldots, a_n \rangle$ is a sequence of activities ordered by their occurrence. A trace where activity a precedes activity b is denoted by $\langle a, b \rangle$, while an empty trace is denoted by ϵ. The alphabet Σ is the set of activities in L. With $\textsc{Start}(L)$ and $\textsc{End}(L)$, we denote the set of activities at the start or end of traces in the log and with $\textsc{start}(L, a)$ and $\textsc{end}(L, a)$ how often activity a is a start or end activity in L. $\mathcal{V}$ denotes the set of unique trace variants, the sequence of activities in a trace. We use the silent activity τ to denote the absence of an activity.

A follows relation exists between two activities in a log if there is a trace in which one is followed by the other. The common directly-follows relationship $a \nrightarrow b$ holds if $\exists \sigma \in L | \sigma = \langle \ldots, a, b, \ldots \rangle$ and the eventually-follows relation $a \nrightarrow^+$ holds if $a \nrightarrow b \vee \exists \sigma \in L | \sigma = \langle \ldots, a, \ldots, b, \ldots \rangle$. From these relations, we derive the directly-follows-graph (DFG) and respectively the eventually-follows-graph (EFG). The nodes in these graphs represent the activities of a log while the edges represent the presence of follows relations. The edge weights, denoted as $|a \nrightarrow b|$, indicate the number of times a relation is observed in the log.

Process trees Process trees are tree-structured process models where leaf nodes are labelled with elements from $\Sigma \cup \{\tau\}$, and inner nodes are labelled with an operator $\oplus$. These trees guarantee translatability into sound workflow nets when turned into Petri nets. Without loss of generality, we focus on binary process trees, where each operator node has exactly two children [9].

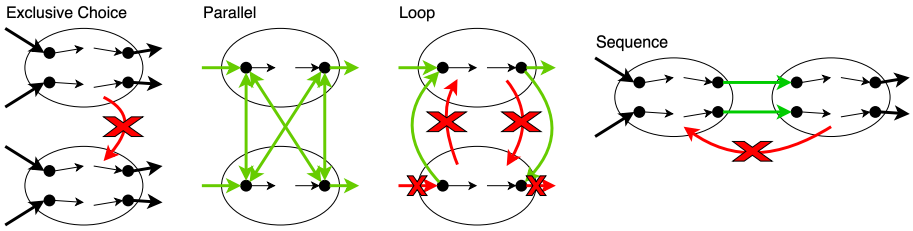

Fig. 1. Visual representations of binary IM cut profiles

The tree leaves are labeled with either an activity $a \in \Sigma$ or τ, representing the execution of an activity or the absence thereof. The operators for inner nodes consist of $\{\rightarrow, \times, ||, \circlearrowleft\}$. The sequence operator $\rightarrow$ mandates that the left child sub-tree be executed before the right. The exclusive choice operator $\times$ allows only one of the children to be executed. The parallel operator $||$ signifies the independent parallel execution of the children. Lastly, the loop operator $\circlearrowleft$ requires an initial execution of the left child, the body part, and permits its repetition after each execution of the right child, the redo part. For formal semantics of process trees, we refer to [9].

Inductive Miner The Inductive Miner (IM) framework uses a recursive approach to construct a process tree from an event log. At each recursion IM finds a maximal non-trivial cut. This involves determining an operator and partition Σ_1 and Σ_2 of activities in the log and subsequently splitting the log into L_1 and L_2 to continue the recursion on the sublogs. This mining decision of operator and partition is called a cut. The base cases of the algorithm generate a τ or a single activity leaf if the event log contains no activities or only a single class of activities.

If no cut and no base case are found, a fall-through method is applied. These methods include structures like τ-skips, enabling optional behaviour, and τ-loops, allowing for arbitrary repetition of behaviour. A flower model is the usual last resort, which is a structure enabling the arbitrary execution of activities in the log, which often comes at the cost of precision. Each cut must adhere to its definitions as detailed in [10], with visual representations provided in Figure 1. A *sequence cut* requires that eventually follows relations are only present from the first part to the second, not the other way around.

$$\forall_{a \in \Sigma_0, b \in \Sigma_1} a \nrightarrow^{+} b \wedge b \nrightarrow^{+} a \tag{1}$$

For an *exclusive choice* cut, activities are partitioned as such that no directly follows relations exist between the partitions:

$$\forall_{a \in \Sigma_0, b \in \Sigma_1} a \not\rightarrow b \wedge b \not\rightarrow a \tag{2}$$

In a *parallel cut* the partitions must be fully interconnected, and must both contain start and end activities.

$$\forall_i \Sigma_i \cap \textsc{Start}(L) \neq \emptyset \wedge A_i \cap \textsc{End}(L) \neq \emptyset \tag{3}$$
$$\forall_{a \in \Sigma_0, b \in \Sigma_1} a \nrightarrow b \wedge b \nrightarrow a \tag{4}$$

Lastly, a *loop cut* restricts start and end activities to the body part. With directly follows relations between the partitions only being permitted from the end activities of the body to the start of the redo part, and from end activities of the redo part to the start activities of the body part.

$$\Sigma_0 \supseteq \textsc{Start}(L) \cup \textsc{End}(L) \tag{5}$$
$$\forall_{a \in \Sigma_0} \exists_{b \in \Sigma_1} b \nrightarrow a \Rightarrow a \in \textsc{Start}(L) \tag{6}$$
$$\forall_{a \in \Sigma_0} \exists_{b \in \Sigma_1} a \nrightarrow b \Rightarrow a \in \textsc{End}(L) \tag{7}$$

4 Opt-IM-II-st

OptIMIIst extends IM by applying a custom fallthrough function. The implementation of it is outlined in Algorithm 1 and visualised in Figure 2. First, the $\textsc{findcuts}_{\textsc{OptIMIIst}}$ function finds potential cuts for each operator based on relaxations of the cut definitions, creating a set of cut candidates. In a second step, evaluateCut evaluates the candidates, including τ-loop and τ-skip, by estimating the local fitness and precision of the candidates selecting the best to apply. OptIMIIst as such always returns a suitable cut and maintains the same rediscoverability guarantees as IM, a fact demonstrated regardless of the fall-through methods employed [9].

4.1 FindCuts for each Operator

The $\textsc{findcuts}_{\textsc{OptIMIIst}}$ function returns optimal partitions for each of the four operators $\{\rightarrow, \times, ||, \circlearrowleft\}$. To achieve this, we formulate and solve a separate Integer Linear Programs (ILP) for each operator. Shared structures between all ILPs are the binary decision variables $x_a \forall a \in \Sigma$, each variable determining whether an

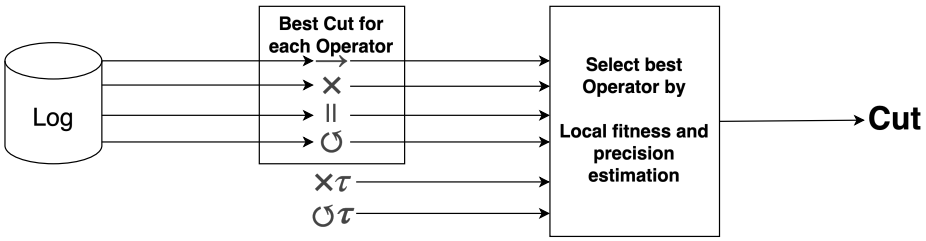

Fig. 2. Top-Level overview of OptIMIIst

Algorithm 1 Opt-IM-II-st Fallthrough

function $\textsc{fallthrough}_{\textsc{OptIMIIst}}(L)$
  $C \leftarrow \{\textsc{findcuts}_{\textsc{OptIMIIst}}(L)\} \cup \{(\times, L, \emptyset), (\circlearrowleft, L, \emptyset)\}$
  $(\oplus_{res}, L_{res1}, L_{res2}) \leftarrow \underset{(\oplus, L_1, L_2) \in C}{\operatorname{argmax}} \textsc{evaluateCut}((\oplus, L_1, L_2))$
  return $\oplus_{res}(\textsc{im}(L_{res1}), \textsc{im}(L_{res2}))$
end function

activity is assigned to the left ($x_a = 1$) or right leaf of the operator node, and constraints (9) and (10), ensuring that neither partition is empty. The formal cut definitions from [10] are relaxed in the ILPs by constructing objective functions that maximize conforming behaviour while penalizing behaviour that violates the cut definitions.

Starting with the *sequence cut*, definition (1) requires that the eventually follow relations between partitions are unidirectional, from part 1 to part 2. To relax this constraint, we allow behaviour in both directions but penalise any flow from part 2 to part 1 in the objective function (8).

$$\text{maximize} \quad \sum_{a \in \Sigma} \sum_{b \in \Sigma} |a \twoheadrightarrow^{+} b| \cdot (x_b - x_a) \tag{8}$$

$$\text{subject to} \quad \sum_{a \in \Sigma} x_a \geq 1 \tag{9}$$

$$\sum_{a \in \Sigma} (1 - x_a) \geq 1 \tag{10}$$

$$x_a \in \{0, 1\} \qquad \forall a \in \Sigma \tag{11}$$

Definition (2) of the *exclusive choice cut* allows for no behaviour between the partitions. In a relaxed version of the definition, we aim to minimize the total flow between the partitions in the objective function (12). We introduce a new set of binary auxiliary variables $z_{a,b}$ in (15), which through (13) and (14) function as a relaxed logical exclusive or between x_a and x_b, meaning $z_{a,b}$ is set to 1 if the activities are in different partitions.

$$\text{minimize} \quad \sum_{a \in \Sigma} \sum_{b \in \Sigma} |a \twoheadrightarrow b| \cdot z_{a,b} \tag{12}$$

$$\text{subject to} \quad (9) \wedge (10) \wedge (11)$$

$$z_{a,b} \geq x_a - x_b \qquad \forall a, b \in \Sigma \tag{13}$$

$$z_{a,b} \geq x_b - x_a \qquad \forall a, b \in \Sigma \tag{14}$$

$$z_{a,b} \in 0, 1 \qquad \forall a, b \in \Sigma \tag{15}$$

For the *parallel cut*, definition (4) requires the partitions to be interconnected, which relaxed translates into maximizing the flow between the two partitions. We use the $z_{a,b}$ variables as a xor with (13), (14), (17) and (18). According to definition (3), we also expect both partitions to contain start and end activities.

To include this in the objective function (16) we additionally aim for an even distribution of these activities between the partitions. We introduce a new variable $b \in \mathbb{N}$, calculated in (19) [4] as the difference of the number of start and end activities between the parts, subtracting it from the objective.

$$\begin{aligned}
\text{maximize} \quad & \sum_{a \in \Sigma} \sum_{b \in \Sigma} z_{a,b} \cdot |a \twoheadrightarrow b| - b && (16)\\
\text{subject to} \quad & (9) \wedge (10) \wedge (11) \wedge (13) \wedge (14) \wedge (15) \\
& z_{a,b} \leq x_a + x_b \quad \forall a, b \in \Sigma && (17)\\
& z_{a,b} \leq 2 - x_a - x_b \quad \forall a, b \in \Sigma && (18)\\
& b \geq |\Sigma_{a \in \Sigma} \textsc{End}(L, a) - 2 \cdot \Sigma_{a \in \Sigma} x_a \cdot \textsc{End}(L, a)| && (19)\\
& + |\Sigma_{a \in \Sigma} \textsc{Start}(L, a) - 2 \cdot \Sigma_{a \in \Sigma} x_a \cdot \textsc{Start}(L, a)| \\
& b \in \mathbb{N} && (20)
\end{aligned}$$

The *loop cut* allows flow from the end activities of the body part to the start activities of the redo part and from the redo part to the start activities of the body part, as defined in (6) and (7). This flow is maximised in the objective function (21), using auxiliary variables $v_{a,b}$ and $w_{a,b} \in \{0, 1\}$ set through constraints (22) - (31), using $\mathcal{X}$ as the indicator function in some of the constraints. The auxiliary variables re_a and rs_a are used for activities which are start and end activities of the redo part. Any partition-crossing behaviour that does not originate from the end activities or does not end in the start activities is considered a violation of (6) and (7) and is represented by the auxiliary variables $f_{a,b} \in \{0, 1\}$, set in constraint (32), and are subtracted from the objective function. Additionally as defined in (5), any start or end activities of the log present in the redo part are also in violation and subtracted from the objective.

$$\begin{aligned}
\text{maximize} \quad & \sum_{a \in \Sigma} \sum_{b \in \Sigma} (v_{a,b} \cdot |a \twoheadrightarrow b| + w_{a,b} \cdot |a \twoheadrightarrow b|) && (21)\\
& - \sum_{a \in \text{START}(L) \cup \text{END}(L)} (1 - x_a) \\
& - \sum_{a \in \Sigma} \sum_{b \in \Sigma} f_{a,b} \cdot |a \twoheadrightarrow b| \\
\text{subject to} \quad & 9 \wedge 10 \wedge 11 \\
& \text{rs}_a \leq (1 - x_a) && \forall a \in \Sigma \quad (22)\\
& \text{re}_a \leq (1 - x_a) && \forall a \in \Sigma \quad (23)\\
& \text{rs}_b \geq (x_a - x_b) \cdot a \twoheadrightarrow b && \forall a \in \textsc{End}(L) \quad (24)\\
& \text{re}_a \geq (x_b - x_a) \cdot a \twoheadrightarrow b && \forall b \in \textsc{Start}(L) \quad (25)\\
& w_{a,b} \geq \text{rs}_b + \mathcal{X}_{\textsc{End}(L)}(a) - 1 && \forall a, b \in \Sigma \quad (26)\\
& w_{a,b} \leq \text{rs}_b && \forall a, b \in \Sigma \quad (27)
\end{aligned}$$

[4] In a practical implementation the absolute value would need to be split into multiple constraints, making use of additional auxiliary variables.

$$w_{a,b} \leq \mathcal{X}_{\text{END}(L)}(a) \qquad \forall a, b \in \Sigma \qquad (28)$$
$$v_{a,b} \geq \text{re}_a + \mathcal{X}_{\text{START}(L)}(b) - 1 \qquad \forall a, b \in \Sigma \qquad (29)$$
$$v_{a,b} \leq \text{re}_a \qquad \forall a, b \in \Sigma \qquad (30)$$
$$v_{a,b} \leq \mathcal{X}_{\text{START}(L)}(b) \qquad \forall a, b \in \Sigma \qquad (31)$$
$$f_{a,b} \geq |(x_a - x_b)| \cdot a \twoheadrightarrow b - w_{a,b} - v_{a,b} \qquad \forall a, b \in \Sigma \qquad (32)$$
$$f_{a,b}, v_{a,b}, w_{a,b} \in \{0, 1\} \qquad \forall a, b \in \Sigma \qquad (33)$$
$$\text{rs}_a, \text{re}_a \in \{0, 1\} \qquad \forall a \in \Sigma \qquad (34)$$

4.2 Operator Selection by local Fitness and Precision Estimation

When approaching the task of choosing the best cut from the candidate set, the objective values produced by the ILPs are incompatible due to the different underlying measurements. Instead OptIMIIst uses local precision and fitness estimation unique for each operator. For fitness we measure the amount of behaviour in the log that is not replayable after applying the cut, while for precision, we propose frequency-based approaches. Constructing expected probability distributions of the model and comparing them with the observed behaviour using the mean absolute error (MAE), calculated as:

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

For the fitness of the *sequence cut*, violating edges are those arcs in the directly follows graph that cross from part 2 to part 1. Those not being replayable after applying the cut. We define the fitness as the fraction of the frequency of those violating edges to the frequency of all crossing edges. For the precision measure, we assume that the probability of transitioning from any $a \in \text{END}(L_1)$ to any $b \in \text{START}(L_2)$ is uniform. And calculate the MAE between these probabilities with the observed transition probabilities in the log.

$$F_{seq} = 1 - \frac{\Sigma_{a \in \Sigma_2} \Sigma_{b \in \Sigma_1} |a \twoheadrightarrow b|}{\Sigma_{a \in \Sigma_2} \Sigma_{b \in \Sigma_1} |a \twoheadrightarrow b| + \Sigma_{a \in \Sigma_1} \Sigma_{b \in \Sigma_2} |a \twoheadrightarrow b|} \qquad (35)$$
$$P_{seq} = 1 - \text{MAE}(P_{Observed}, P_{Expected}) \qquad (36)$$

For the *exclusive choice cut*, fitness is calculated as the fraction of existing directly follows edges to the total possible edges between activities from different partitions. While precision is always 1, as an exclusive choice cut does not introduce any additional behaviour.

$$F_{xor} = 1 - \frac{\Sigma_{a \in \Sigma_2} \Sigma_{b \in \Sigma_1} a \twoheadrightarrow b + \Sigma_{a \in \Sigma_1} \Sigma_{b \in \Sigma_2} a \twoheadrightarrow b}{2 \cdot |\Sigma_1| \cdot |\Sigma_2|} \qquad (37)$$

The fitness of a *parallel cut* is always 1, as it guarantees that all possible interleaved combinations of traces from the sub-logs can be replayed. For precision, we estimate the expected number of variants by calculating $2^{\text{AVGLEN}(\mathcal{V}_1)+\text{AVGLEN}(\mathcal{V}_2)}$,

where $\text{AVGLEN}(\mathcal{V}_1)$ and $\text{AVGLEN}(\mathcal{V}_2)$ are the average trace lengths of the sub-logs $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively. This expected number of variants, V_{exp}, is then compared to the observed number of variants, $|\mathcal{V}|$, to compute the precision as:

$$P_{and} = \frac{|\mathcal{V}|}{V_{\text{exp}}} \tag{38}$$

For the *loop cut* fitness, start and end activities will not be replayable after applying the cut. As such, we calculate the fitness as the fraction of such activities in part 2. For precision we assume that both partitions are independent of each other and the transition probabilities from end activities of the body part to the start activities of the redo part as well as the transition probabilities from the end activities of the redo part to the start activities of the body part are uniform. We calculate the MAE between these probabilities and the observed transition probabilities in the log.

$$F_{loop} = 1 - \frac{|\text{START}(L_b)| + |\text{END}(L_b)|}{|\text{START}(L)| + |\text{END}(L)|} \tag{39}$$

$$P_{loop} = 1 - \frac{\text{MAE}(P_{obsA\to B}, P_{expA\to B}) + \text{MAE}(P_{obsB\to A}, P_{expB\to A})}{2} \tag{40}$$

To get a single comparison measure we calculate the F1-score, defined as $F_1 = 2 \times \frac{fitness \times precision}{fitness + precision}$, and select the candidate with the highest F1-score to apply. For the τ-skip and τ-loop, we use proxy measures from [7].

5 Evaluation

In this section, we evaluate OptIMIIst against other state-of-the-art discovery algorithms. Additionally, we discuss some limitations of OptIMIIst.

Setup To compare OptIMIIst with other discovery algorithms, we select three existing algorithms. We use IMf as a baseline comparison, with a filter setting of 0.2, AIM, and the ILP-Miner. The latter an approach outside the IM framework. For our evaluation, we utilize public event logs from the BPI Challenges of 2012[5], 2013[6], 2017[7], and 2020[8], as well as the Sepsis log[9]. We split each log into a train/test split with 80% of cases in train and 20% of cases in the test split. Models were mined on the train logs and alignment-based fitness [8] was evaluated on the test logs. Additionally, alignment-based precision [1] of the models was measured with the full logs. We also assessed the size of the resulting Petri nets (number of places, transitions, and arcs), the number of activities in the models, recorded the frequency of fallthrough occurrences in the OptIMIIst

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

Table 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

and the runtime of each miner. To mitigate the effect of randomness in the splits, we repeated each experiment five times and averaged the results.

OptIMIIst was implemented in Python, using Gurobi[10] to construct and solve the ILPs. All tests were conducted on macOS 14.5 with an M3-Max processor, with RAM usage consistently remaining below 8GB.

Results Table 1 presents the results. The fallthrough mechanism was triggered at least once for all logs, and more than five times for all logs except the BPIC 2013 and Sepsis logs. OptIMIIst generally outperforms the IMf in both terms of precision and fitness. While falling short of AIM archiving similar fitness scores, but consistently worse precision. The ILP-Miner, while achieving perfect fitness on most logs, produces models that are very large, in case of BPIC 2020_{TPD} almost nine times larger than those generated by the IM based Miner, and exhibits the worst precision across all logs. Although the precision issue could

[10] Gurobi Optimizer Reference Manual - https://www.gurobi.com

be partially addressed by manual optimisation the filter parameters, this manual step would likely also reduce fitness.

Comparing the model size of the IM based miners a similar picture as for fitness and precision can be seen. OptIMIIst produces, with one exception, smaller models than IMf but larger ones than AIM. Looking at the number of activities present in the model the reason for that is clearly visible. While OptIMIIst never excluded any activity, AIM performs excessive activity filtering on the activity level, trading information loss for increased precision and simpler models, which leads to a high loss of activities in logs. Like for the BPIC 2020 logs in which AIM filters almost 50% of activities, with in BPIC 2020_{DD} only exhibiting 7 of the original 17 activities. An issue not present in OptIMIIst, upholding similar fitness results to AIM.

Comparing runtimes, IMf is unsurprisingly the fastest miner as it only performs filtering. While OptIMIIst is generally faster than AIM, but slower than ILP-Miner. From a soundness perspective, the models of the IM framework miners are all sound while only 30% of the ILP-Miners models being sound.

Limitations While our evaluation results show a promising performance of OptIMIIst, some limitations should be acknowledged. One key concern is the computational complexity of using ILPs for optimization. Although our evaluation showed promising runtimes, the approach may have non-polynomial complexity, and therefore, we cannot guarantee efficient computation in all cases.

The optimality of our approach is confined to the local decisions made during the mining process and the relaxation assumptions embedded in the ILPs and estimator. Since OptIMIIst does not employ backtracking, there is no guarantee that an optimal local cut will lead to the globally best mining decision.

6 Conclusion

This paper introduced a novel twofold approach for identifying locally optimal process trees. Our method finds locally optimal activity partitions for each operator, while handling infrequent and incomplete behaviour. Utilizing a fitness and precision estimator to select the most suitable cut. We evaluated OptIMIIst by demonstrating its competitive performance in terms of precision, fitness and simplicity compared to the existing algorithms AIM, IMf and ILP-Miner. Notably it was competitive without excessive activity filtering. As future work, OptIMIIst could be extended with explicit filtering of activities in conjunction with the cut optimization routines. Furthermore, the ILPs could be used to find more complex structures like long-distance-dependencies and duplicated activities, for example by incorporating external data.

References

1. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Measuring precision of modeled behavior. Inf. Syst. E Bus. Manag. **13**(1) (2015)

2. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L.: Split miner: Discovering accurate and simple business process models from event logs. In: 2017 IEEE International Conference on Data Mining, ICDM 2017. IEEE Computer Society (2017)
3. 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)
4. Bergenthum, R.: Prime miner - process discovery using prime event structures. In: ICPM 2019. IEEE (2019)
5. Brons, D., Scheepens, R., Fahland, D.: Striking a new balance in accuracy and simplicity with the probabilistic inductive miner. In: ICPM (2021)
6. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. Int. J. Cooperative Inf. Syst. **23**(1) (2014)
7. van Detten, J.N., Schumacher, P., Leemans, S.J.J.: An approximate inductive miner. In: ICPM 2023 (2023)
8. van Dongen, B., Carmona, J., Chatain, T., Taymouri, F.: Aligning modeled and observed behavior: A compromise between computation complexity and quality. In: AiSE. Cham (2017)
9. Leemans, S.J.J.: Robust Process Mining with Guarantees - Process Discovery, Conformance Checking and Enhancement, LNBIP, vol. 440. Springer (2022)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - A constructive approach. In: Petri Nets. LNCS, vol. 7927 (2013)
11. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM Workshops - BPM. LNBIP, vol. 171 (2013)
12. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: Petri Nets. LNCS (2014)
13. Leemans, S.J.J., Tax, N., ter Hofstede, A.H.M.: Indulpet miner: Combining discovery algorithms. In: On the Move to Meaningful Internet Systems. OTM 2018 Conferences. Cham (2018)
14. Mannel, L.L., van der Aalst, W.M.P.: Finding complex process-structures by exploiting the token-game. In: Petri Nets. Lecture Notes in Computer Science, vol. 11522. Springer (2019)
15. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE **77**(4) (1989)
16. Sani, M.F., van Zelst, S.J., van der Aalst, W.M.P.: Repairing outlier behaviour in event logs. In: Business Information Systems. Lecture Notes of Business Information Systems, vol. 320 (2018)
17. Solé, M., Carmona, J.: Encoding process discovery problems in SMT. Softw. Syst. Model. **17**(4) (2018)
18. Weijters, A.J., Van der Aalst, W.M.: Rediscovering workflow models from event-based data using little thumb. Integrated Computer-Aided Engineering **10**(2) (2003)
19. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: Petri Nets. Berlin, Heidelberg (2008)