Striking a new Balance in Accuracy and Simplicity with the Probabilistic Inductive Miner

Dennis Brons, Roeland Scheepens
UiPath Process Mining
Eindhoven, The Netherlands
{dennis.brons,roeland.scheepens}@uipath.com

Dirk Fahland
Eindhoven University of Technology
The Netherlands
d.fahland@tue.nl

***Abstract*—Numerous process discovery techniques exist for generating process models that describe recorded executions of business processes. The models are meant to generalize executions into human-understandable modeling patterns, notably parallelism, and enable rigorous analysis of process deviations. However, well-defined models with parallelism returned by existing techniques are often too complex or generalize the recorded behavior too strongly to be trusted in a practical business context. We bridge this gap by introducing the Probabilistic Inductive Miner (PIM) based on the Inductive Miner framework. PIM compares in each step the most probable operators and structures based on frequency information in the data, which results in block-structured models with significantly higher accuracy. All design choices in PIM are based on business context requirements obtained through a user study with industrial process mining experts. PIM is evaluated quantitatively and in an novel kind of empirical study comparing users' trust in discovered model structures. The evaluations show that PIM strikes a unique trade-off between model accuracy and model complexity, that is conclusively preferred by users over all state-of-the-art process discovery methods.**

I. Introduction

Discovering a process model from an event log is a central step in any process mining analysis in an industrial setting. The discovered model summarizes the data and aids analysts in understanding the executed process and in distinguishing main behavior and deviations.

Research in the last 20 years contributed numerous algorithms for discovering process models supporting concurrency in syntax and semantics using human-understandable modeling notations [1, 2]. The models have to be sound [3], have high *fitness* and *precision*, and be simple in *structure* [4, 2]; best state-of-the-art techniques according to a recent benchmark [2] strike different trade-offs and do not meet all criteria. For instance, the Split Miner [5] returns models with high fitness and precision that tend to have a more complex structure and may be unsound; the Inductive Miner [6] returns sound, block-structured models with high fitness and simpler structure, but low precision.

Industrial applications still favor directly-follows graphs (DFGs) which is anecdotally attributed to their simple semantic concepts. For industrial analysts, process models with formal modeling notations are harder to understand and reason about than DFGs once models reach a certain complexity [7]. However, the perceived simplicity of DFGs is countered by their inability to describe processes with concurrency, leading to false statistics and wrong insights regarding performance and deviating behavior [8] while analysts require correct information.

In this paper we (1) investigate and empirically evaluate which balance of quality properties in a process model with concurrency is preferred by analysts in an industrial setting, and (2) address the problem of identifying and developing a corresponding process discovery algorithm.

Preferred quality properties. We answered (1) through computational requirements and a user study. As process models of an event log form a pareto-front along fitness, precision, and simplicity, models with a simpler structure have lower fitness to the data [4]. A low-fitting model can be visually augmented with the "missing" behavior by visually overlaying deviating paths computed from alignments [9]. However, computing alignments is expensive [10] hindering quick interactive exploration of data. *Visual Alignments* [11] are an approximation of alignments for the purpose of visualization that can be computed in linear time on block-structured BPMN models with only XOR- and AND-gateways (see [12] for an example). As block-structured models are also easier to understand [7], we identify requirement (**R0**) *The discovered model must be block-structured with only XOR- and AND-gateways.*

To further answer (1), we conducted a *Delphi user study* for which we prepared 9 fragments of 3 real-life event logs. For each fragment we manually created between 2 and 8 alternative block-structured process models differing in structural complexity, fitness, and precision. We asked 6 expert process mining analysts to indicate which models they prefer as representation for each fragment (and why). Preferences and reasons were consolidated in a second round, resulting in the following requirements for process discovery in an industrial context: (**R1**) *The algorithm must have a parameter to control model complexity* to allow the analyst include fewer/more details at the cost of fitness/precision. (**R2**) *The algorithm must produce model structures for which there is significant evidence in the data.* Specifically, parallelism, loops, and skipping of activities should only be shown when occurring so "frequently" that an analyst does not doubt the algorithm's choice.

Algorithm development. To develop an algorithm that satisfies (R0-R2), we chose the Inductive Miner framework

2021 3rd International Conference on Process Mining (ICPM 2021), available at https://doi.org/10.1109/ICPM53251.2021

as it ensures block-structured models. However, all existing IM algorithms, strike the wrong trade-off to satisfy (R2), specifically IMf [6] filters infrequent behaviors using a very basic, local heuristic. The IMc [13] algorithm instead uses a probabilistic model to infer missing behavior. We chose to combine ideas from IMf and IMc to develop a Probabilistic IM algorithm, we called *PIM*, that can handle infrequent behavior.

We recall preliminaries about the IM framework and IMc in Sect. II. We then introduce the Probabilistic Inductive Miner (PIM) in Section III. We implemented PIM in the UiPath Process Mining platform and evaluated PIM quantitatively and empirically (see Sect. IV): we found that PIM strikes a unique balance between fitness, precision, and model complexity: PIM models achieve high precision while sacrificing fitness, and PIM consistently returns models with lower complexity than other algorithms. In a novel empirical evaluation, we ask users to indicate their trust in structures discovered by different techniques; PIM's unique balance is preferred by users. We present our concluding remarks in Section V.

II. Background

We recall event logs, process trees, and the IM framework and the IMf and IMc algorithms. A *trace* $\sigma \in \Sigma^*$ is a finite sequence of activity names Σ observed for one process execution; an *event log* $L \in \mathcal{B}(\Sigma^*)$ is multi-set of traces. The *directly-follows graph* (DFG) $\nrightarrow (L)$ of L has nodes Σ; an edge from $a \in \Sigma$ to $b \in \Sigma$, written $a \nrightarrow b$, iff b directly follows a in some trace $\langle \dots a, b, \dots \rangle \in L$. Correspondingly, the *indirectly-follows graph* (IFG) $\nrightarrow^* (L)$ has an edge $a \nrightarrow^* b$ iff b strictly indirectly follows a in some trace $\langle \dots a, \dots, b, \dots \rangle \in L$. The *frequency* $|a \nrightarrow b|$ and $|a \nrightarrow^* b|$ are the number of occurrences of b in L which directly and indirectly follow some a, respectively, e.g., in $\langle a, a, b, c, b, b, a, b \rangle$, $|a \nrightarrow b| = 2$ (1st, 4th b) and $|a \nrightarrow^* b| = 4$ (all 4 b). $Start(L)$ and $End(L)$ denote the sets of first and last activities in the traces in L; with correspondingly defined frequencies.

The following log L_0 serves as our running example: $L_0 = [\langle a, b, c, e \rangle, \langle a, c, b, f \rangle, \langle a, b, c, d, c, b, e \rangle^2, \langle a, c, b, d, b, c, f \rangle, \langle a, c, b, d, b, c, d, b, c, f \rangle, \langle a, g \rangle^9, \langle a, g, c, g \rangle]$; Fig. 1(left) visualizes $\nrightarrow (L_0)$; arcs without source/target node indicate $Start(L_0)$ and $End(L_0)$.

A *process tree* (PT) is an abstract representation of a sound, block-structured workflow net [14]. A PT is a tree where each leaf node is an activity $a \in \Sigma$ and every non-leaf node is an operator $\oplus \in \{\times, \rightarrow, \wedge, \circlearrowleft\}$. Each sub-tree defines a block of $\times$ (exclusive choice), $\rightarrow$ (sequence), $\wedge$ (interleaved parallelism), or $\circlearrowleft$ (loop) over its children; first child of $\circlearrowleft$ is the loop body which can be repeated after executing any of the other "redo" children. For example, log L_0 was generated from the PT in Fig. 1(right) with some deviations, e.g., $\langle a, g, c, g \rangle$.

The *Inductive Miner* (IM) [15] defines a framework for recursively a PT from an event log in 3 steps. (1) BaseCase(L) checks whether L has a trivial structure for which a trivial solution can be found (e.g., L contains just a single activity). Otherwise, (2) FindCut(L) identifies a *cut*: a PT operator

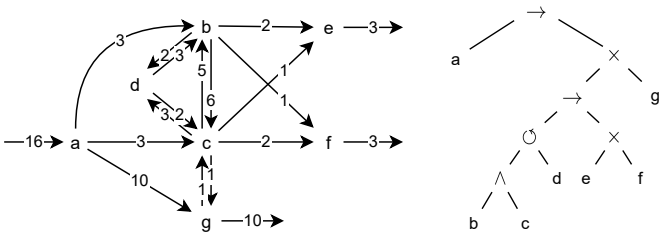


Fig. 1: Directly-follows relation of L_0 (left) and process tree (right) from which L_0 was generated with some deviations.

$\oplus \in \{\times, \rightarrow, \wedge, \circlearrowleft\}$ that best describes best the relation between n partitions $\Sigma_1, ..., \Sigma_n$ of the activities Σ in L (e.g., a sequence or a choice over n blocks of activities). If a cut is found, (3) SplitLog(L) splits L according to the identifier operator and partitions into $L_1, \dots, L_n$, e.g., partition the set of traces ($\times$), sequentially split each trace ($\rightarrow$, $\circlearrowleft$), or project each trace ($\wedge$), in a way that maximizes fitness, and recursively call IM$(L_1), \dots,$ IM(L_n).

IM infrequent (IMf) [6] detects an n-ary cut in $\nrightarrow (L)$ by trying to *top-down* partition Σ according to an $\times$, $\rightarrow$, $\wedge$, and $\circlearrowleft$ operator (in this order) based on the structure of $\nrightarrow (L)$. To detect cuts in the presence of deviations, IMf filters out relations from $\nrightarrow (L)$ occurring relatively less often than the strongest relation at an activity. If still no cut is found, IMf returns the flower model which fits any log over Σ. But IMf's top-down approach and some design choices in filtering lead to IMf too often not finding the most likely cut in the presence of slightly "too much" deviating behavior. This results in low precision on real-life data [2].

In contrast, *IM incomplete* (IMc) [13] detects a *binary* cut in $\nrightarrow (L)$ and $\nrightarrow^* (L)$ in a *bottom-up fashion*: it computes for any pair (a_1, a_2) of activities the probability $p_\oplus(a_1, a_2)$ that a_1 and a_2 are related by $\oplus \in \{\times, \rightarrow, \wedge, \circlearrowleft\}$. Then, it constructs an SMT problem to search for a partition $\Sigma = \Sigma_1 \cup \Sigma_2$ and operator $\oplus$ where the aggregated probabilities of all pairs $p_\oplus(a_1, a_2), a_1 \in \Sigma_1, a_2 \in \Sigma_2$ is maximal. IMc always finds the most likely cut and requires no flower model, but the $p_\oplus(a_1, a_2)$ are computed under the assumption of incomplete (missing) behavior in L and cannot filter out deviating behavior, making it inapplicable on real-life data.

Of the other state-of-the-art algorithms [2], *ETM* [14] uses a genetic search over PTs to maximize fitness, precision, generalization and simplicity; it often finds block-structured models with better precision and fitness than IM but at very high running times. *Heuristic mining techniques* [16, 17, 18] determine the most likely (frequent) relations between activities through quotients over $\nrightarrow (L)$ and $\nrightarrow^* (L)$ and then derive split/join logic by counting succeeding/preceding activities in traces, but cannot guarantee block-structured process models; restructuring into blocks [5] fails on most real-life logs.

In the following, we combine the ideas of IMc to detect cuts through bottom-up computation of the most likely operator between activities with the idea of quotients over $\nrightarrow (L)$ and $\nrightarrow^* (L)$ in logs with deviations used in heuristic miners.

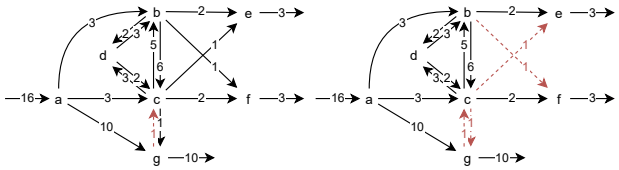


Fig. 2: Filtering $\nrightarrow (L)$ of Fig. 1 by IMf with $f = 0.15$ (left) and our method with $f = 97\%$ (right).

III. The Probabilistic Inductive Miner

We now propose a new instantiation of the IM framework (cf. Sect. II) called *Probabilistic Inductive Miner* (PIM) with the following pseudo-code.

function PIM(L, f)
  $bc \leftarrow$ BaseCase(L)
  if $bc \neq \square$ **then**
    return $bc(L)$
  else
    $\nrightarrow (L)^f, \nrightarrow^* (L)^f \leftarrow$ Filtering($\nrightarrow (L), \nrightarrow^* (L), f$)
    $\oplus(\Sigma_1, \Sigma_2) \leftarrow$ FindCut($\nrightarrow (L)^f, \nrightarrow^* (L)^f$)
    $L_1, L_2 \leftarrow$ SplitLog($L, \oplus(\Sigma_1, \Sigma_2)$)
    return $\oplus$ (PIM(L_1, f), PIM(L_2, f)
  end if
end function

We first discuss our new definitions of filtering the DFG (Filtering) in Sect. III-A, FindCut in Sect. III-B, SplitLog and BaseCase in Sect. III-C. We apply PIM on our running example L_0 in Sect. III-D before discussing some implementation details in Sect. III-E.

A. Filtering

The IMf algorithm [6] introduced filtering infrequent edges from $\nrightarrow (L)$ when no cut can be found in $\nrightarrow (L)$. IMf filters $\nrightarrow (L)$ locally, by removing all outgoing edges $a \nrightarrow b$ with $|a \nrightarrow b| < f \cdot \max_{x \in \Sigma} |a \nrightarrow x|$. This filters $\nrightarrow (L)$ non-uniformly, as shown in Fig. 2(left) where edges $c \nrightarrow g$ and $g \nrightarrow c$ of same frequency are partially filtered (filtered edges in red). This impairs IMf's ability to control model complexity (BCR1) in a uniform way.

We chose to adopt percentile-based filtering for a more uniform control of model complexity. Filtering($\nrightarrow (L)$, $\nrightarrow^* (L), f$) sorts all edges in $\nrightarrow (L)$ and $\nrightarrow^* (L)$ by their frequency, retains only the top $f\%$ of frequent (in)directly follows edges and discards all others; disconnected nodes are removed. Filtering Fig. 1(left) in this way removes edges uniformly as shown in Fig. 2(right). We use this method to parameterize the model complexity (BCR1).

B. Cut Detection

For FindCut, we extend the basic idea of IMc [13] with principles of Heuristic Mining. First, we compute for each pair (a, b) of activities a score $s_\oplus(a, b) \in [0, 1]$ how likely it is that a and b are related by $\oplus \in \{\times, \rightarrow, \wedge, \circlearrowleft\}$ in L. For a given partition $\Sigma = \Sigma_1 \cup \Sigma_2$ of the activities in L, we then can compute an aggregated score $s_\oplus(\Sigma_1, \Sigma_2)$, i.e., the likelihood that the activities Σ_1 and Σ_2 form two blocks related by $\oplus$ in L. We then determine the cut $(\oplus, \Sigma_1, \Sigma_2)$ with $s_\oplus(\Sigma_1, \Sigma_2)$ maximal.

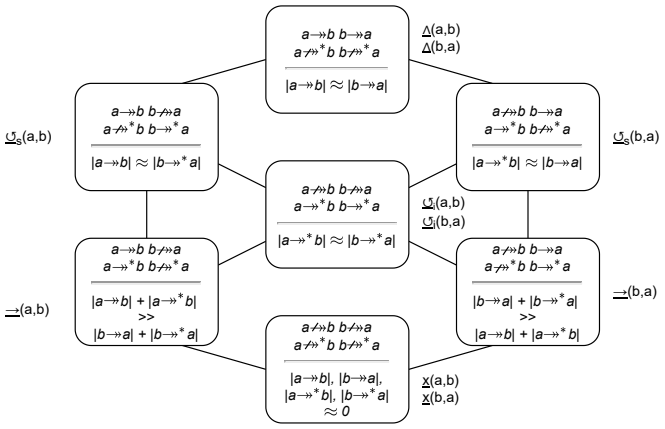


Fig. 3: Activity Relation Lattice.

1) Activity Relation Scores: The lattice in Fig. 3 illustrates the basic idea for determining the likelihood $s_\oplus(a, b)$. The IMc [13] introduced the mutually exclusive conditions over $\nrightarrow$ and $\nrightarrow^*$ shown above the horizontal line to describe when a and b are related by a particular operator in the absence of deviations, e.g., a and b are related by $\times$ if neither follows the other (indirectly). We translated these into the fuzzy conditions over how often a and b (in)directly follow each other shown below the horizontal line, e.g., a and b are related by $\times$ if they rarely follow each other (indirectly).

To quantify the likelihood that $\oplus$ relates a and b we defined the following formulas; as for IMc [13] for $\circlearrowleft$ we distinguish between $\circlearrowleft_s$ (a directly follows b in a loop) and $\circlearrowleft_i$ (a indirectly follows b in a loop). In addition to $|a \nrightarrow b|$ and $|a \nrightarrow^* b|$ we write $|a|$ for the number of times a occurs in L.

Definition 1 (activity relation scores)**.**

$$s_\times(a,b) = \frac{|a| - (|a \nrightarrow b| + |b \nrightarrow a| + |a \nrightarrow^* b| + |b \nrightarrow^* a|)}{|a|}/2 + \frac{|b| - (|a \nrightarrow b| + |b \nrightarrow a| + |a \nrightarrow^* b| + |b \nrightarrow^* a|)}{|b|}/2$$

$$s_\rightarrow(a,b) = \frac{|a \nrightarrow b| + |a \nrightarrow^* b| - (|b \nrightarrow a| + |b \nrightarrow^* a|)}{|a \nrightarrow b| + |a \nrightarrow^* b| + |b \nrightarrow a| + |b \nrightarrow^* a| + 1}$$

$$s_\wedge(a,b) = \min\left(\frac{|a \nrightarrow b|}{|b \nrightarrow a| + 1}, \frac{|b \nrightarrow a|}{|a \nrightarrow b| + 1}\right)$$

$$s_{\circlearrowleft_s}(a,b) = \min\left(\frac{|a \nrightarrow b|}{|b \nrightarrow^* a| + 1}, \frac{|b \nrightarrow^* a|}{|a \nrightarrow b| + 1}\right)$$

$$s_{\circlearrowleft_i}(a,b) = \min\left(\frac{|a \nrightarrow^* b|}{|b \nrightarrow^* a| + 1}, \frac{|b \nrightarrow^* a|}{|a \nrightarrow^* b| + 1}\right)$$

In contrast to IMc [13], the scores are *not* probabilities with $\sum_\oplus s_\oplus(a, b) = 1$ but heuristics measuring which of the fuzzy conditions shown in Fig. 3 is most likely to hold.

For $\times$, we compare how often a and b occur alone, i.e., $|a|$ and $|b|$, with how often they occur together, i.e., $|a \nrightarrow b| + |b \nrightarrow a| + |a \nrightarrow^* b| + |b \nrightarrow^* a|$). If either only a or b occurs in a trace, then $|a \nrightarrow b| = \ldots = |b \nrightarrow^* a| = 0$, and $s_\times(a, b) = 1$; otherwise the second term grows and $s_\times(a, b)$ approaches 0. In L_0, $s_\times(b, g) = 1$.

For $\rightarrow$, we adopt the directional dependency heuristic of the Structured Heuristic Miner [5]. a is before b in a sequence if the evidence for "a precedes b" ($|a \nrightarrow b| + |a \nrightarrow^* b|$) significantly exceeds the evidence for "b precedes a" ($|b \nrightarrow a| + |b \nrightarrow^* a|$). To keep our scores within the 0 to 1 range, we normalize the difference and truncate negative values to 0. For any activities a and b, $s_{\rightarrow}(a,b) = -s_{\rightarrow}(b,a)$; $s_{\rightarrow}(a,b)$ is maximal if $|b \nrightarrow a| = |b \nrightarrow^* a| = 0$. In L_0, $s_{\rightarrow}(a,g) = .92$.

For $\wedge$, we adopt the heuristics of IMc [13] and Heuristic Miner [19]. The more equal $|a \nrightarrow b|$ and $|b \nrightarrow a|$ are, the more evidence is in L for a parallel relation, and the higher is $s_{\wedge}(a, b)$. We take the minimum of $\frac{|a \nrightarrow b|}{|b \nrightarrow a|+1}$ and $\frac{|b \nrightarrow a|}{|a \nrightarrow b|+1}$ to ensure $s_{\wedge}(a,b)$ lies between 0 and 1. In L_0, $s_{\wedge}(b,c) = \min(5/7, 6/6) = .71$.

For $\circlearrowleft$, we compute two scores to distinguish indirect loop relation ($s_{\circlearrowleft_i}(a,b)$) from entering/exiting the redo part of the loop ($s_{\circlearrowleft_s}(a,b)$). If a is in the loop body and $a \nrightarrow b$ enters the redo part of a loop (b is a redo activity), then we see $|a \nrightarrow b|$ (entering the redo) as often as $|b \nrightarrow^* a|$ (returning from the redo to the body), i.e., their quotient in $s_{\circlearrowleft_s}(a,b)$ is close to 1. If $a \nrightarrow b$ exits the redo part, then the converse $s_{\circlearrowleft_s}(b,a)$ will be close to 1 (see Fig. 3). An indirect loop relation $\circlearrowleft_i\ (a,b)$ is likely when a and b indirectly follow ($|a \nrightarrow^* b|$) and precede ($|b \nrightarrow^* a|$) with similar frequency.

The activity scores have the property that if L is directly-follows complete and $\oplus$ (mostly) holds between a and b in L, then $s_{\oplus}(a,b) > s_{\otimes}(a,b)$ for all operators $\otimes \neq \oplus$. If $\times$ holds, then $0 = |a \nrightarrow b| = \ldots = |b \nrightarrow^* a|$ and $s_{\otimes}(a,b) = 0$ for $\otimes \neq \times$. If $\rightarrow$ holds, then $|b \nrightarrow a| = |b \nrightarrow^* a| = 0$, thus $s_{\otimes}(a, b) = 0$ for $\otimes \in \{\wedge, \circlearrowleft_i, \circlearrowleft_s\}$. Also $|a \nrightarrow b| + |a \nrightarrow^* b| = |b|$ in this case (c.f. Sect. II), thus $s_{\times}(a,b) < .5$. If $\wedge$ holds, then $|a \nrightarrow b| \approx |b \nrightarrow a|$ thus $s_{\times}(a,b) \approx 0$ and $s_{\rightarrow}(a,b) \approx 0$. In the same way $\circlearrowleft_i$ and $\circlearrowleft_s$ exclude $\times$ and $\rightarrow$. Distinguishing $\wedge$ from $\circlearrowleft_i$ and $\circlearrowleft_s$ requires counting repetitions [20] which we do when aggregating activity scores.

2) Aggregated Scores for Cuts: We now aggregate the activity relation scores $s_{\oplus}(a,b)$ to sets $s_{\oplus}(\Sigma_1, \Sigma_2)$ of activities $\Sigma = \Sigma_1 \cup \Sigma_2$ so that we can search for cuts $(\oplus, \Sigma_1, \Sigma_2)$ with $s_{\oplus}(\Sigma_1, \Sigma_2)$ being maximal. We write $S(\oplus, R) = [s_{\oplus}(a, b) \mid (a,b) \in R]$ for the bag of scores of activity pairs R.

For L having no deviations, IMc defined $s_{\oplus}(\Sigma_1, \Sigma_2)$ as the *average* $\mu(S(c)) = (\sum_{s \in S(c)} s)/|S(c)|$ of all activity relation scores $S(c) = S(\oplus, \Sigma_1 \times \Sigma_2)$ in cut c, with a special case for $\oplus = \circlearrowleft$ [6]. However, deviations in L may give a few pairs (a,b) a biased score $s_{\otimes}(a,b) > s_{\oplus}(a,b)$ which could lead to a "wrong" average $\mu(S(\otimes, \Sigma_1' \times \Sigma_2')) > \mu(S(\oplus, \Sigma_1 \times \Sigma_2))$ although $s_{\otimes}(x,y) < s_{\oplus}(x,y)$ for most pairs (x,y). We therefore introduce a *correction term or factor* to obtain a lower score $s_{\otimes}(\Sigma_1', \Sigma_2') < \mu(S(\otimes, \Sigma_1', \Sigma_2'))$ when we see evidence of such wrong bias. We first present the formal definitions and provide a full example in Sect. III-D.

For $\oplus \in \{\times, \rightarrow\}$, we know from Sect. III-B1 that $s_{\oplus}(a, b)$ is low iff $s_{\otimes}(a,b), \oplus \neq \otimes$ is high (a and b related by $\otimes$ and not by $\oplus$). Thus, a high average over $s_{\oplus}(a,b), (a, b) \in \Sigma_1 \times \Sigma_2$ is falsely biased towards $\oplus$ if some pairs (a,b) have a significantly lower score $s_{\oplus}(a,b)$ than other pairs. We therefore use as correction term for $\times$ and $\rightarrow$ the *standard deviation* $\sigma(S)$ over a set S of activity relation scores.

Definition 2 (aggregate score for $\times$, $\rightarrow$)**.** *Let* $\Sigma_1 \cup \Sigma_2 = \Sigma$*;* $\oplus \in \{\times, \rightarrow\}$. $c = (\oplus, \Sigma_1, \Sigma_2)$ *is an* $\oplus$-cut *with score* $s_{\oplus}(\Sigma_1, \Sigma_2) = \mu(S(c)) - \sigma(S(c))$.

Suppose we have two cuts $c_1 = (\times, \Sigma_1, \Sigma_2)$ and $c_2 = (\rightarrow, \Sigma_1', \Sigma_2')$ with $S(c_1) = [.6, .6, .7]$ and $S(c_2) = [.9, .8, .4]$ the set of activity relation scores for c_1 and c_2. Then $s_{\times}(\Sigma_1, \Sigma_2) = .63 - .05 = .58$ and $s_{\rightarrow}(\Sigma_1', \Sigma_2') = .7 - .21 = .48$, i.e., c_1 has the higher aggregate score although $\mu(c_1) < \mu(c_2)$.

For $\oplus \in \{\wedge, \circlearrowleft\}$, we know from Sect. III-B1 and Def. 1 that (1) if $s_{\otimes}(a,b), \otimes \in \{\times, \rightarrow\}$ is low then (a,b) can be in $\wedge$ or $\circlearrowleft$ relation, and vice versa, and that (2) we cannot reliably distinguish parallel behavior from loop behavior using only binary activity relations.

Distinguishing $\wedge$ from $\circlearrowleft$ requires explicitly counting repetitions [20], which we achieve as follows. We write $|L|$ for the number of traces and $||L||$ for the total number of events in log L. If L has no repetition (no loop) then each trace in L contain each $a \in \Sigma$ at most once and $||L|| \leq |L| \cdot |\Sigma|$. For instance, $L_1 = [\langle a,b\rangle, \langle b,a\rangle]$ has $||L|| = 4 \leq 2 \cdot 2 = |L| \cdot |\Sigma|$ whereas $L_2 = [\langle a,b\rangle, \langle b,a,b\rangle]$ has $5 \not\leq 2{\cdot}2$. Thus, $r(L) = |L| / \frac{||L||}{|\Sigma|} < 1$ if L has a loop, e.g., $r(L_1) = 1$ and $r(L_2) = 2/(5/2) = 0.8$. We thus can use $r(L)$ as correction factor to reduce the $\wedge$ score (presence of loops), which we bound to $[0;1]$ by $\min(r(L), 1)$.

Definition 3 (aggregate score for $\wedge$)**.** *Let* $\Sigma_1 \cup \Sigma_2 = \Sigma$. *Then* $c = (\wedge, \Sigma_1, \Sigma_2)$ *is a* $\wedge$*-cut with* score $s_{\wedge}(\Sigma_1, \Sigma_2) = \mu(S(c)) \cdot \min(r(L), 1)$.

Finally, IMc [13] computes the score for $\circlearrowleft$ as average over $\circlearrowleft_i$ and $\circlearrowleft_s$ as follows. In a loop, the redo part Σ_2 is entered from/exited to body Σ_1 at $S_2, E_2 \subseteq \Sigma_2$, respectively. Pairs (a,b) directly entering/exiting the redo part are scored using $\circlearrowleft_s$, while all other (indirect) pairs are scored using $\circlearrowleft_i$ (see Fig. 3). As $r(L) \approx 1$ or $r(L) \geq 1$ only indicates absence of $\circlearrowleft$ but not presence of $\wedge$, we use the inverse $(1 - \min(r(L), 1))$ to boost a $\circlearrowleft$-score by reinforcing the presence of loops.

Definition 4 (aggregated score for $\circlearrowleft$)**.** *Let* $\Sigma_1 \cup \Sigma_2 = \Sigma$ *and* $S_2, E_2 \subseteq \Sigma_2$. *Then* $c = (\circlearrowleft, \Sigma_1, \Sigma_2, S_2, E_2)$ *is a* $\circlearrowleft$-cut *defining sets* $enter = End(L) \times S_2$, $exit = E_2 \times Start(L)$, *and* $indirect = (\Sigma_1 \times \Sigma_2) \setminus (enter \cup exit)$.

The scores for c *are* $S(c) = S(\circlearrowleft_s, enter \cup exit) \cup S(\circlearrowleft_i, indirect)$. *The* aggregated score *for* c *is* $s_{\circlearrowleft}(\Sigma_1, \Sigma_2, S_2, E_2) = \mu(S(c)) + (\mu(S(c)) \cdot (1 - \min(r(L), 1)))$.

The correction term $\ldots + (\mu(S(c)) \cdot (1 - \min(r(L), 1))$ boosts the loop score relative to the inverse $(1 - \min(r(L), 1))$ which is high when $r(L) \ll 1$, i.e., L shows many repetitions. In this way, cuts $c_1 = s_{\wedge}(\Sigma_1, \Sigma_2)$ and $c_2 = s_{\circlearrowleft}(\Sigma_1', \Sigma_2', \ldots)$ with $S(c_1) \approx S_(c_2)$ due roughly equally many directly and indirectly-following activity pairs can be distinguished through either $\min(r(L), 1)$ boosting $s_{\wedge}(\Sigma_1, \Sigma_2)$ or $(1 - \min(r(L), 1)$ boosting $s_{\circlearrowleft}(\Sigma_1', \Sigma_2')$; see Sect. III-D for an example.

3) Cut finding: A naïve method for FINDCUT computes all activity relation scores $s_{\oplus}(a,b)$ for all $(a,b) \in \Sigma \times \Sigma$ and then

exhaustively searches for a partition $\Sigma_1 \cup \Sigma_2$ with $s_{\oplus}(\Sigma_1, \Sigma_2)$ or $s_{\circlearrowleft}(\Sigma_1, \Sigma_2, E_2, S_2), E_2, S_2 \subseteq \Sigma_2$ being maximal, which costs $O(|\Sigma|^2) + O(4^{|\Sigma|})$. Heuristics in $\nrightarrow$ and $\nrightarrow^*$ allow to significantly prune the search space for $\Sigma_1, \Sigma_2, E_2, S_2$ but are omitted for space limitations. Note that this method always returns a cut (possibly with a low score) while IMf may not find a cut and resort to a flower model [6].

C. Log Splitting, Recursion, Base Cases, Skipping

Following the IM framework, we next split the log L according to the found cut $c = (\oplus, \Sigma_1, \Sigma_2)$ into L_1 and L_2. As c may not completely fit L due to deviations, we base SPLITLOG on IMf's [6] method which filters from L_1 and L_2 events that deviate from $(\oplus, \Sigma_1, \Sigma_2)$. Filtering may result in empty traces $\langle\rangle \in L_i$, denoting that the behavior in L_i can also be skipped. Then PIM is invoked on L_1 and L_2, returning a left and right sub-tree under $\oplus$ (see start of Sect. III).

We use the IM base cases, with some extensions regarding the handling of empty behavior in a (sub)log.

- SINGLE ACTIVITY ($|\Sigma| = 1$). When the sublog L contains only a single activity, that activity is returned as a leaf node of the process tree.
- NO ACTIVITIES ($|\Sigma| = 0$). If the sublog L does not contain any activities, a silent activity (τ) is returned as leaf node of the process tree.
- SKIP SUBLOG When the sublog contains n empty traces $\langle\rangle^n \in L$, a skip sub-tree $\times(\tau, \text{PIM}(L \setminus \{\langle\rangle\}))$ is returned as explained next.

IMf's BASECASE always filters $\langle\rangle^n$ from L and returns $\times(\tau, \text{IMF}(L \setminus \{\langle\rangle\}))$ if $n \geq |L| \cdot f$ wrt. threshold f. This either forgets that some empty behavior has been seen or results in many τ-skips across all subtrees which lowers precision [2]. We delay generating $\times(\tau, \ldots)$ as follows. If $n \leq .5 \cdot |L|$, we keep all empty traces in L but ignore them when computing $\rightarrow$, $\rightarrow^*$, and $r(L)$. Upon log splitting, L_1 and L_2 both get all $\langle\rangle^n$ empty traces and filtering may add further ones. Only when we accumulated $n > .5 \cdot |L|$ empty traces, i.e., the majority of the behavior in L is skipping, SKIP SUBLOG introduces the $\times(\tau, \ldots)$ which addresses (R2).

D. Example

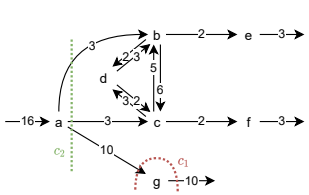


Fig. 4: Cut c_1 vs c_2

Consider L_0 with filtered DFG in Fig. 2(right) and cuts $c_1 = (\times, \{a, b, c, d, e, f\}, \{g\})$ and $c_2 = (\rightarrow, \{a\}, \{b, c, d, e, f, g\})$ in Fig. 4. The mean scores (as in IMc) are $\mu(c_1) = .86 > \mu(c_2) = .79$, because g has strong $\times$ relations to b, c, d, e, f. However, $s_{\times}(a, g) = .16 \ll 1$ shows that some other relation holds between a and g and $\mu(c_1)$ is falsely biased towards $\times$; this shows in $\sigma(c_1) = .29$. In contrast, $s_{\rightarrow}(a, y)$ is high for *all* $y \in \{b, c, d, e, f, g\}$ due to $a \nrightarrow y$ or $a \nrightarrow^* y$ (no other activity is likely), and $\sigma(c_2) = .13$. Thus, $s_{\times}(\{a, b, c, d, e, f\}, \{g\}) = .54$ and $s_{\rightarrow}(\{a\}, \{b, c, d, e, f, g\}) = .67$, making c_2 the more likely cut.

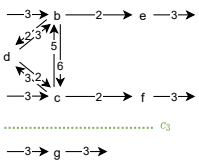


Fig. 5: Cut c_3

After log splitting and applying BASECASE on $L_1 = [\langle a \rangle^{16}]$, PIM finds $c_3 = (\times, \{b, c, d, e, f\}, \{g\})$ on $\nrightarrow$ and $\nrightarrow^*$ of L_2 in Fig. 5. Log splitting filters c from $L_3 = [\langle g \rangle, \langle g, c, g \rangle$ BASECASE returns g. On $L_4 = [\langle b, c, e \rangle, \langle c, b, f \rangle, \langle b, c, d, c, b, e \rangle^2, \langle c, b, d, b, c, f \rangle, \langle c, b, d, b, c, d, b, c, f \rangle]$.

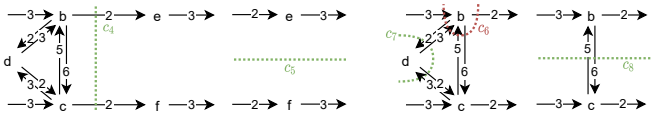


Fig. 6: Cuts c_4-c_8 show effect of correction factor in Def.2-4.

Subsequently, PIM calculates and chooses cut $c_4 = (\rightarrow, \{b, c, d\}, \{e, f\})$, see Fig. 6a. For the right-hand side, PIM finds $c_5 = (\times, \{e\}, \{f\})$ with base cases for e and f, see Fig. 6b, respectively. In $\nrightarrow$ and $\nrightarrow^*$ of $L_5 = [\langle b, c \rangle, \langle c, b \rangle, \langle b, c, d, c, b \rangle^2, \langle c, b, d, b, c \rangle, \langle c, b, d, b, c, d, b, c \rangle]$ of the left-hand side (Fig. 6c) we see the effect of correction term $r(L_5)$ for distinguishing $\wedge$ and $\circlearrowleft$.

Consider cuts $c_6 = (\wedge, \{b\}, \{c, d\})$ and $c_7 = (\circlearrowleft, \{b, c\}, \{d\})$in Fig. 6c. Without correction, $\mu(S(c_6)) = 1.0$ and $\mu(S(c_7)) = .79$. As $r(L_5) = .75$, the correction factors yield $s_{\wedge}(\{b\}, \{c, d\}) = \mu(S(c_6)) \cdot r(L_5) = .75$ and $s^m_{\circlearrowleft}(\{b, c\}, \{d\}) = \mu(S(c_7)) \cdot (2 - .75) = .99$, and PIM returns c_7. After log splitting, PIM finds $c_8 = (\wedge, \{b\}, \{c\})$ on $L_6 = [\langle b, c \rangle^6, \langle c, b \rangle^5]$ at the "correct" location (see Fig. 6d). Applying BASECASE then results in the process tree of Fig. 1.

E. Complexity

The running time complexity of PIM depends on the size of the event log L, and the size of its alphabet Σ as follows. Per recursion step, BASECASE is executed once, constructing and filtering $\nrightarrow$ and $\nrightarrow^*$ takes $O(|\Sigma|^2)$, FINDCUT takes $O(4^{|\Sigma|})$ (see Sect. III-B3), and SPLIT LOG takes $O(|L|)$, decreasing the size of the alphabet in each sublog by at least one. Thus, there are at most Σ invocations of PIM resulting in running-time complexity of $O(|\Sigma| \cdot (1 + |\Sigma|^2 + 4^{|\Sigma|} + |L|)) = O(|\Sigma| \cdot (4^{|\Sigma|} + |L|))$. As discussed in Sect. III-B3, heuristics over $\nrightarrow^+$ and $\nrightarrow^*$ reduce the search space for FINDCUT significantly as measured in our experiments.

IV. EVALUATION

We implemented PIM in the UiPath Process Mining platform and compared it to the state-of-the-art on a standard benchmark [2] and on additional data more similar to regular use cases encountered in industrial practice (IV-A). In a novel empirical evaluation, we tested which structures produced by different algorithms are preferred by end users (IV-B).

A. Quantitative Evaluation

Setup. The existing benchmark [2] uses a set of non-synthetic event logs to comparatively evaluate 7 automated process discovery methods of which Evolutionary Tree Miner

Log Name	*Total Traces*	*Distinct Traces (%)*	*Total Events*	*Distinct Events*
BPIC12	13,087	33.4	262,200	36
$\text{BPIC13}_{\text{cp}}$	1,487	12.3	6,660	7
$\text{BPIC13}_{\text{inc}}$	7,554	20.0	65,533	13
BPIC14_{f}	41,353	36.1	369,485	9
$\text{BPIC15}_{1\text{f}}$	902	32.7	21,656	70
$\text{BPIC15}_{2\text{f}}$	681	61.7	24,678	82
$\text{BPIC15}_{3\text{f}}$	1,369	60.3	43,786	62
$\text{BPIC15}_{4\text{f}}$	860	52.4	29,403	65
$\text{BPIC15}_{5\text{f}}$	975	45.7	30,030	74
BPIC17_{f}	21,861	40.1	714,198	41
RTFMP	150,370	.2	561,470	11
SEPSIS	1,050	80.6	15,214	16
BPIC14	46,616	48.5	466,737	39
$\text{BPIC17}_{\text{LC}}$	31,509	53.0	1,202,267	66
Invoice	24,450	1.6	133,452	15
P2P	616,717	15.5	5,583,650	46

TABLE I: Statistics of the public event logs, extracted from [2], followed by the additional event logs.

(ETM) [14], IMf [6] and Split Miner (SM) [17], outperformed the other methods. We therefore compare PIM to ETM, IMf, and SM on the public benchmark event logs of [2], see Tab. I. Model accuracy is measured by fitness, precision, and their F-score; simplicity is measured by size, control-flow complexity (CFC), structuredness, and soundness, see [2].

However, the benchmark is not representative of industrial workloads: (1) Most event logs were prefiltered to reduce complexity [2] while techniques in industrial practice face unfiltered logs, (2) industrial logs are larger, and (3) the benchmark lacks certain process types found in practice. We thus included 4 additional unfiltered datasets: 2 public datasets BPIC14 and $\text{BPIC17}_{\text{LC}}$ (distinguish activity life-cycles), and 2 proprietary datasets (Invoice and P2P); see Tab I.

We ran PIM twice: (1) bounding the naïve FINDCUT with complexity $O(4^{|\Sigma|})$ (Sect. III-B3) to only find cuts over the 30 most frequent activities to adhere to the 4 hour computation limit set in [2] (PIM_{30}); (2) using fast heuristics in FINDCUT for which no activity bound was required; in both cases $f = 99.5\%$ to filter only the most infrequent behavior. On the 4 additional logs, we ran IMf and SM with default parameters, but omit EMT due to its excessive running times.

Results. PIM and PIM_{30} discovered sound, block-structured models on all event logs while SM returned unsound models on BPIC14 and $\text{BPIC17}_{\text{LC}}$.

Figure 7 shows fitness, precision, f-score, size, CFC, and running times for ETM, IMf, SM as reported in [2] and for PIM_{30} and PIM for all datasets; see [12] for individual measures reported. The scatter plots in Fig. 8 and Fig. 9 visualize fitness, precision, and F-score against size and CFC (normalized against the largest size/CFC measured per log).

On the benchmark event logs, PIM sacrifices fitness for increased precision (11/12 logs better than IM, 9/12 same or better than SM, 5/12 highest precision) and F-score (10/12 better than IMf, 4/12 highest, and 5/12 scoring 2nd best close to SM) while improving size (8/12 better than IMf, 4/12 better than SM) and CFC (10/12 better than IMf, 6/12 same or better than SM). PIM_{30} sacrifices fitness even more and further

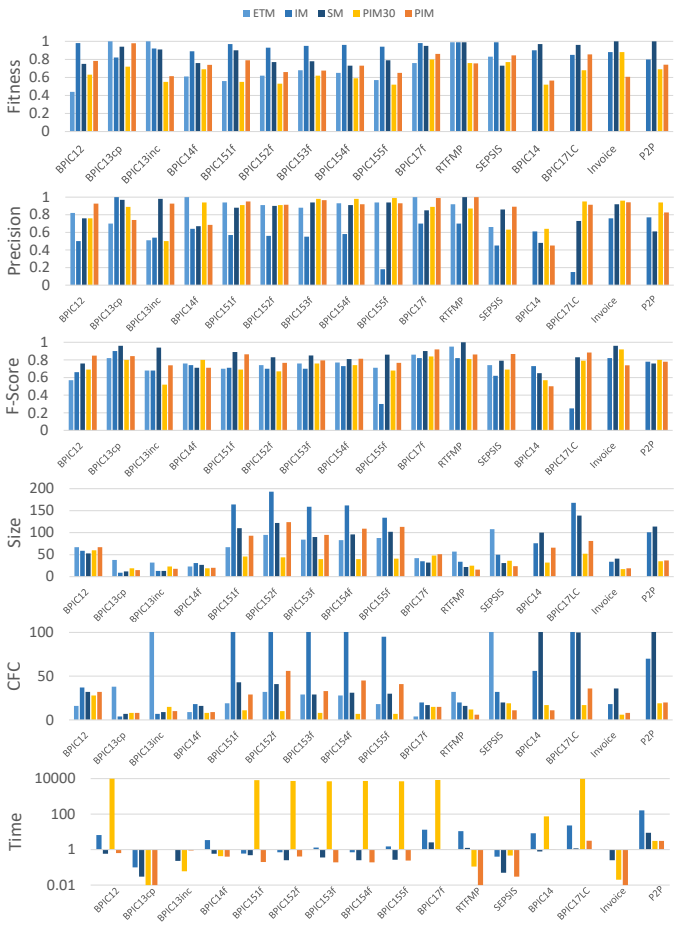


Fig. 7: Fitness, precision, f-score, size, CFC, and running time for ETM, IMf, SM [2], PIM_{30}, and PIM

reduces size and CFC due to activity filtering. PIM consistently shows higher fitness (12/12) and higher F-score (11/12) than PIM_{30} but also shows higher precison on 7/12 logs. PIM's and PIM_{30}'s results are more similar to those of ETM wrt. accuracy than to IMf and SM but finds larger models as well as smaller models than ETM does. The scatter plots in Fig. 8 show that PIM strikes a novel balance in the pareto-front having more results with high precision and low size/CFC (top-left corner) compared to IMf and SM while the overall F-score for is comparable to all other techniques. PIM_{30} either finds a solution within a second (5/12 logs) or takes several hours to complete whereas PIM completes in less than a second for all data sets (fastest for 11/12) due to the heuristics in FINDCUT.

On the 4 additional datasets, SM achieves highest fitness and PIM sacrifices fitness most. PIM_{30} (and PIM) clearly outperforms IMf and SM in precision, size, and CFC on all (3/4) datasets, yielding F-scores comparable to IMf and SM (see Fig. 7); PIM models are slightly larger and less accurate than PIM_{30} models. The scatter plots in Fig. 9 reinforce the prior finding even stronger: PIM_{30} and PIM reach a novel area in the pareto-front of quality criteria by striking a much better balance in precision and simplicity not reached by other

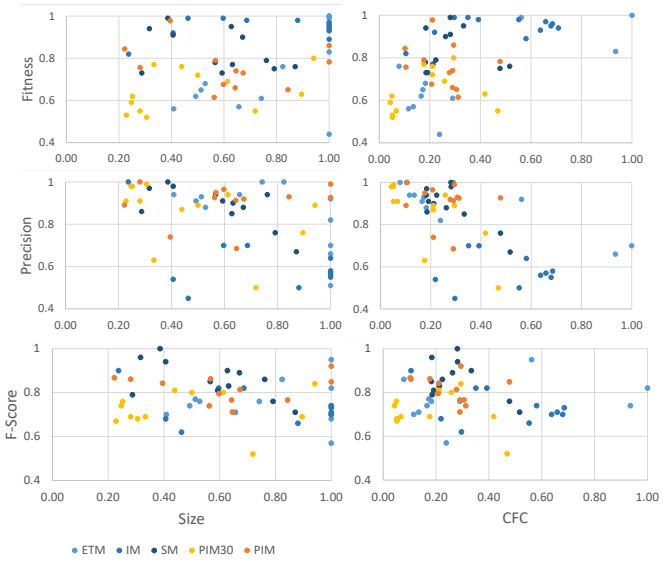


Fig. 8: Fitness, precision, f-score vs normalized size and CFC of ETM, IMf, SM, PIM_{30}, and PIM on benchmark.

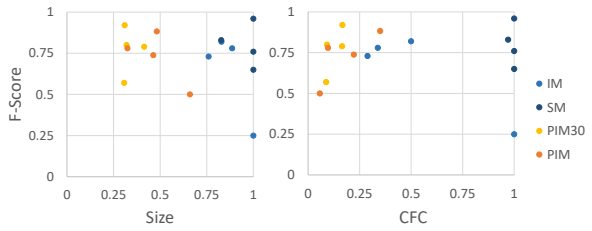
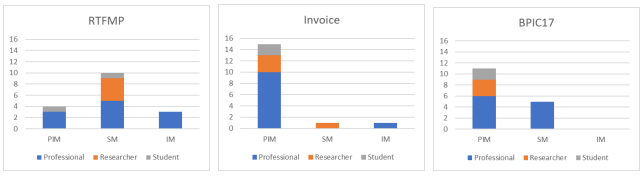


Fig. 9: F-score vs normalized control-flow complexity of IMf, SM, PIM_{30} on additional benchmark.

techniques while retaining comparable F-scores.

Parameter sensitivity. To verify that this is not due to parameter choices, we evaluated how accuracy and size change when changing IMf's, SM's, and PIM's parameter for filtering infrequent behavior on the 4 additional datasets. Increasing filtering for SM traded fitness for precision which results in near-constant accuracy and in up to 50% fewer edges for models returned by SM and corresponding reduction of CFC. However SM size and CFC remained significantly higher than that of PIM with $f = 99.5\%$. Increasing filtering for IMf lead to unpredictable changes in accuracy and size; size and complexity always remained above PIM with $f = 99.5\%$. Thus SM and IMf cannot reach the particular spot of simplicity vs accuracy reached by PIM. Lowering PIM's filtering parameter f allowed to reduce model size and CFC down to the 2 most frequent activities; model size and complexity thereby falls off in a negative exponential curve as many infrequent edges in $\nrightarrow$ and $\nrightarrow^*$ that contribute to complexity appear to be "equally infrequent" and get filtered out together when lowering f. This confirms that PIM's models remain simpler also under filtering parameters. Moreover, both the effective filtering parameter and the option to consider only the k most frequent activities in cut detection allow to control model detail and complexity, satisfying (R1).



(a) RTFMP (b) Invoice (c) $\text{BPIC17}_{\text{cp}}$

Fig. 10: Favorite model choices made by the participants of the questionnaire

B. Empirical Evaluation

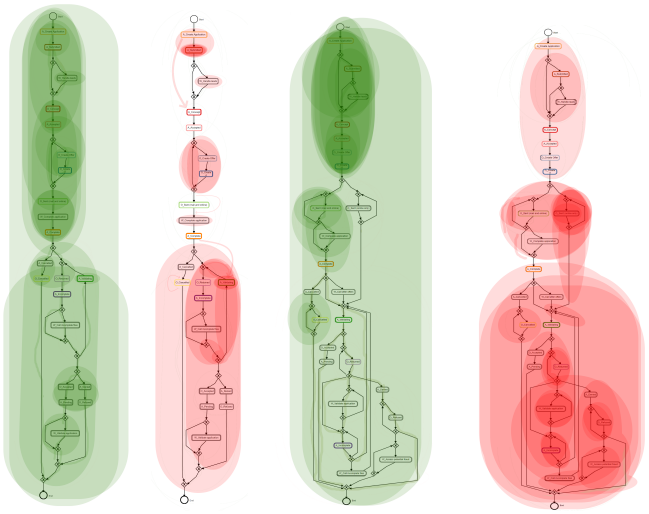
Setup. We performed an empirical evaluation to test whether the particular spot of high accuracy and simplicity taken by PIM indeed achieves R2: that the models produced by PIM only show structures for which there is significant evidence in the data. We provided participants (11 professionals, 4 researches, 2 students) with the following. (1) A visualization of the most frequent trace variants with activities color-coded as generated by Prom's "Explore Event Log" visualizer that could fit legibly on one A4 sheet for RTFMP (99% of the log), Invoice (94% of the log), and $\text{BPIC17}_{\text{cp}}$ ($\text{BPIC17}_{\text{LC}}$ filtered to only "complete" events, 55% of the log). (2) Automatically laid-out diagrams of BPMN translations of the respective models produced by IMf, SM, and PIM (setup as in Sect. IV-A), with algorithm anonymized and in random order. (3) Instructions to highlight with pens of two distinct colors all model structures they consider "good" (understand and have sufficient evidence in the data) or "bad" (do not understand or lack evidence). (4) A questionnaire to rank model preference and indicate their reasons. No time limit was given.

Results. Where SM was the most-preferred model for RTFMP (10/17), PIM was preferred most for Invoice (15/17) and $\text{BPIC17}_{\text{cp}}$ (11/17); see Fig. 10. To understand which structures end users trust most, we aggregated the individual participants' highlighting of "good" and "bad" model structures into heat maps. Fig. 11 shows the heat maps for the PIM and SM models of $\text{BPIC17}_{\text{LC}}$; see [12] for all heat maps.

From the heat maps and questionnaires, we observed that the participants generally trusted and understood model structures produced by PIM more than model structures produced by SM and IMf (c.f. Fig. 11b vs. 11d). Models by PIM were sporadically described as representing too little of the data, but never too much of the data. Of the 51 written motivations justifying the participants' model ranking, 30 (58.8%) justified their choice for PIM due to simplicity, clarity, or readability of the models. These results together confirm the particular usefulness of PIM's unique balance in fitness vs precision vs simplicity seen in Sect. IV-A, satisfying R2.

V. Conclusion

We combined principles of the IMc [13] algorithm of the IM framework [15] with ideas from heuristic mining [19] to design process discovery algorithm PIM that returns block-structured models of high simplicity and accuracy even on large, unfiltered event logs. PIM strikes a novel balance between higher precision, significantly lower model complexity, and



(a) Good (b) Bad (c) Good (d) Bad

Fig. 11: Markings made on the BPIC17$_{cp}$ models discovered by PIM (a, b) and SM (c, d).

comparable overall accuracy (F-score) compared to existing techniques; thereby reaching a new area in the pareto-front of model quality metrics. Our empirical evaluation confirms that this pareto-front area represents models that are considered user-friendly and accurate for the data which was a central requirement raised by industrial analysts in a Delphi study prior to algorithm development. The models found by PIM are preferred over models produced by other state-of-the-art techniques; deviation analysis is then possible through model enhancement [9] such as visual alignments [11] (c.f. App **??**).

Threats to validity. The standard benchmark [2] only contains filtered variants of real-life event logs changing the nature of the problem, which we offset by additional but fewer unfiltered event logs. All participants of the empirical evaluation were based in the Eindhoven area in the Netherlands possibly introducing bias in preferences. Where IMc [13] computes actual probabilities for operators, PIM only estimates probabilities through heuristics. Moreover, PIM intrinsically approximates behavior when the behavior recorded in the event log is not block-structured, hence accuracy measured partly relates to the degree of structuredness of the underlying process behavior.

Future work. Algorithmically, PIM seems to inherently favor precision over fitness while IMf's and SM's notable quality is ensuring fitness and high fitness/precision balance, respectively. It is worth exploring whether adding further parameters to PIM allows to produce fully fitting and precise models, possibly at the expense of simplicity.

However, in the qualitative evaluation, participants expressed a clear preference for simplicity and accuracy over fitness. This warrants reconsideration of the kinds of models automated process discovery techniques should produce. Process mining may need a focus shift: discovery of fully fitting models is no longer desirable if users do not find them usable.

References

[1] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs," *Information Systems*, 2012.

[2] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated Discovery of Process Models from Event Logs: Review and Benchmark," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 686–705, 2019.

[3] W. M. P. van der Aalst, *Process Mining, Data science in action*. Springer, 2016.

[4] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, "Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity," *Int. J. Cooperative Inf. Syst.*, vol. 23, no. 1, 2014. [Online]. Available: https://doi.org/10.1142/S0218843014400012

[5] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and G. Bruno, "Automated discovery of structured process models from event logs: The discover-and-structure approach," *Data and Knowledge Engineering*, vol. 117, no. April, pp. 373–392, 2018. [Online]. Available: https://doi.org/10.1016/j.datak.2018.04.007

[6] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *BPM 2013 Workshops*, ser. LNBIP, vol. 171. Springer, pp. 66–78.

[7] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7PMG)," *Inf. Softw. Technol.*, vol. 52, no. 2, pp. 127–136, 2010. [Online]. Available: https://doi.org/10.1016/j.infsof.2009.08.004

[8] W. M. [van der Aalst], "A practitioner's guide to process mining: Limitations of the directly-follows graph," *Procedia Computer Science*, vol. 164, pp. 321 – 328, 2019, cENTERIS/ProjMAN/HCist 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050919322367

[9] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Exploring processes and deviations," in *BPM 2014 Workshops*, ser. LNBIP, vol. 202. Springer, 2014, pp. 304–316.

[10] J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking - Relating Processes and Models*. Springer, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-99414-7

[11] B. d. Bie, "Visual Conformance Checking using BPMN," Master thesis, Eindhoven University of Technology, 2019.

[12] D. Brons, R. Scheepens, and D. Fahland, "Striking a new balance in accuracy and simplicity with the probabilistic inductive miner," *CoRR*, vol. abs/2109.06288, 2021. [Online]. Available: http://arxiv.org/abs/2109.06288

[13] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from incomplete event logs," in *Petri Nets 2014*, ser. LNCS, vol. 8489. Springer, 2014, pp. 91–110.

[14] J. C. Buijs, B. F. Van Dongen, and W. M. Van Der Aalst, "Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity," *International Journal of Cooperative Information Systems*, vol. 23, no. 1, 2014.

[15] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs - A constructive approach," in *Petri Nets 2013*, ser. LNCS, vol. 7927. Springer, 2013, pp. 311–329.

[16] S. K. vanden Broucke and J. De Weerdt, "Fodina: A robust and flexible heuristic process discovery technique," *Decision Support Systems*, vol. 100, pp. 109–118, 2017.

[17] A. Augusto, R. Conforti, M. Dumas, and M. L. Rosa, "Split miner: Discovering accurate and simple business process models from event logs," in *Proceedings - IEEE International Conference on Data Mining, ICDM*, vol. 2017-Novem, 2017, pp. 1–10.

[18] A. Augusto, M. Dumas, and M. L. Rosa, "Automated discovery of process models with true concurrency and inclusive choices," in *ICPM 2020 Workshops*, ser. LNBIP, vol. 406. Springer, 2020, pp. 43–56. [Online]. Available: https://doi.org/10.1007/978-3-030-72693-5_4

[19] A. Weijters, W. M. P. van der Aalst, and A. K. A. de Medeiros;, "Process Mining with the HeuristicsMiner Algorithm," 2006.

[20] S. J. Leemans and D. Fahland, "Information-preserving abstractions of event data in process mining," *Knowledge and Information Systems*, 2019.