Repairing Process Models through Simulation and Explainable AI

No Author Given

No Institute Given

Abstract. A process model is one of the main milestones for Business Process Management and Mining. They are used to engage process stakeholders into discussions on how processes should be executed, or are alternatively used as input for Process-aware Information Systems to automate processes. Desirable models need to be precise and only allow legitimate behavior (high precision), while enabling the executions that have been observed (high fitness). Often, models fail to achieve these properties, and need to be repaired. This paper proposes a model-repair framework that compares the behavior allowed by the model with what observed in reality, aiming to pinpoint the distinguishing features. The framework creates a Machine Learning model that discriminates the traces of the real event log w.r.t. those of a synthetic event log obtained via simulation of the process model. Explainable-AI techniques are employed to make the distinguishing features explicit, which are then used to repair the original process model. Our framework has been implemented and evaluated on four processes and various models, proving the effectiveness of enhancing the original process model, in the harmonic means of fitness and precision. Our results are then compared with those obtained through the state of the art, which tend to prefer fitness over precision: the comparison shows that our framework outperforms the literature in balancing fitness and precision.

Keywords: Process-Model Repair · Business Process Simulation · Machine Learning · Explainable AI

1 Introduction

A process model is one of the main milestones for Business Process Management and Mining. They are used to engage process stakeholders into discussions on how processes should be executed, or are alternatively used as input for Process-aware Information Systems to automate processes. Desirable models need to be precise and only allow legitimate behavior, while they should be able to represent the valid executions that have been observed (high fitness).

Process models generally fail to achieve these properties, and need to be repaired. However, fitness and precision are often contrasting: improving one may worsen the other. This paper reports on a novel model-repair technique that tries to improve the harmonic means of fitness and precision.

The key idea is to leverage on business process simulation: the original process model, namely before any repair, is used to generate a set of simulated

traces. These traces are then compared with those of a real event log to highlight the differences. This comparison is based on training a Machine-Learning (ML) model that, given an execution trace, is capable to classify whether it is from the real or simulated event log. The rationale is that, if the process model is characterized by high fitness and precision, the ML model is not capable to determine whether a trace belongs to the real or simulated event log. In other words, the lower is the harmonic means of fitness and precision in a process model, the higher is the accuracy of the ML model. When the ML-model accuracy is high, it is possible to pinpoint which trace characteristics are used for the determination whether the trace is from the real or simulated event log. In this paper, we use the SHAP (SHapley Additive exPlanations) method to pinpoint the differences. These trace characteristics are then employed to automatically improve the process model, aiming to achieve a process model that, once simulated, would produce traces that are indistinguishable from the real event log. This paper introduces different primitives to concretely repair the model.

To demonstrate the effectiveness of our proposed technique, we conducted evaluations on four distinct processes and various models. The results showcase the technique's ability to enhance the original process models, improving on both fitness and precision. The evaluation also compares the resulting repaired models with respect to those obtained by Fahland et al. [9], again highlighting how we achieve a better balance.

The paper is organized as follows. Section 2 reports on preliminary concepts, Section 3 introduces the framework, Section 4 outlines the evaluation and results. Section 5 reports the related works, and finally Section 6 concludes the paper.

2 Preliminaries

This section introduces the preliminary concepts useful to discuss the framework described later.

First, basic definitions of events, traces, and event logs are presented, subsequently, a subsection recalls the notions on Petri nets. A final subsection provides the definitions related to the explainability of a Machine Learning model.

2.1 Events, Traces, Event Logs

Usually process transactional data are collected as event logs. An event log is a collection of traces, where a trace is a sequence of events with their respective attributes. In this paper, we abstract the events as the activities that are executed, and we thus ignore the other event attributes because they are irrelevant for our technique.

Definition 1 (Traces & Event Logs). *Let $\mathcal{A}$ be the set of activity labels. A trace $\sigma = \langle \texttt{start}, e_1, \ldots, e_m, \texttt{end} \rangle \in \mathcal{A}^*$ is a sequence of activities. An event log $\mathcal{L}_\mathcal{A}$ is a multiset of traces, i.e. $\mathcal{L}_\mathcal{A} \in \mathbb{B}(\mathcal{A}^*)$.*

We define a trace to always have the special activities $\texttt{start}, \texttt{end} \in \mathcal{A}$, as this will be useful for defining later the set of trace features. This is not necessary for the framework, and only introduced here to simplify the formalization.

In the following we define a relation function, which will be used to encode the traces for the training of the discriminator models.

Definition 2 (Footprint Relation). *Given a trace* $\sigma = \langle e_0, \ldots, e_m \rangle \in \mathcal{L}_{\mathcal{A}}$*, a pair* $(a, b) \in \mathcal{A} \times \mathcal{A}$ *belongs to the footprint relation* $>_\sigma$ *iff* $\exists\, i \in \{0, \ldots, m-1\}$ *s.t.* $e_i = a$ *and* $e_{i+1} = b$*.*

In the remainder, we use the notation $a >_\sigma b$ to indicate $(a, b) \in >_\sigma$.

2.2 Petri Nets

In this paper we use Petri nets for modeling and representing processes with a set of recorded instances $\mathcal{L}_{\mathcal{A}}$. We opted for Petri nets because they have a simple yet formal syntax and semantics, sufficiently reach to model the aspects relevant for our framework.

Definition 3 (Petri Net). *A Petri net is a triple* (P, T, F)*, where* P *is a finite set of places,* T *is a finite set of transitions, and* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs.*

In the remainder, for any $t \in T$, $t^\bullet = \{p | (t, p) \in F\}$ and ${}^\bullet t = \{p | (p, t) \in F\}$ are the set of places with arcs coming from or going to t, respectively. The state of a Petri net is identified by its current marking:

Definition 4 (Marking). *A marking* m *of a Petri net* (P, T, F) *is a function* $m : P \to \mathbb{N}$ *which assigns for each place* $p \in P$ *the number* $m(p)$ *of tokens at this place. A Petri net system is a Petri net with an initial marking* m^i *and a final marking* m^f*.*

A **Labelled Petri net system** $N = \left(P, T, F, \lambda, m^i, m^f\right)$ consists of a Petri net (P, T, F), a labelling function $\lambda : T \to \mathcal{A} \cup \{\tau\}$ where $\mathcal{A}$ is a set of activities and τ denotes the label of invisible transitions, m^i the initial marking and m^f the final marking. Hereafter, we use the term Petri net to indicate a Labelled Petri net system, without causing ambiguities.

Since Petri nets are used to model business processes, we assume that they are in fact Workflow Nets [2]: there exist an input place $i \in P$ and an output place $o \in P$ that have no incoming or outgoing arcs, respectively, as well as every node is in a path between i and o.

The behaviour of a Petri net $N = \left(P, T, F, \lambda, m^i, m^f\right)$ can be represented as a so-called **reachability graph**, which is a way of representing the states of Petri nets. A reachability graph of a Petri net is a directed graph $G_N = (V, E)$, where:

1. V is the set of nodes where each node $v \in V$ represents a reachable marking (from the initial m^i);
2. E is the set of edges, each of which is a triple (m_i, t, m_j), which indicates that t can fire at marking m_i, leading to marking m_j.

In the remainder, given two activities a and b and a Labelled Petri net system $N = \left(P, T, F, \lambda, m^i, m^f\right)$ with reachability graph $G_N = (V, E)$, $a >_N b$ is used to indicate that, according to N, activity b can be immediately executed after a,

namely E contains a set of edges $\{(m_1, t_1, m_2), \ldots, (m_{n-1}, t_{n-1}, m_n)\}$ such that $\lambda(t_1) = a$, $\lambda(t_{n-1}) = b$ and, for all $1 < k < n-1$, $\lambda(t_k) = \tau$.

This paper aims to obtain sound models. Since we focus on Petri net, we base on the **soundness** criteria introduced in [2], according to which a Petri net $\left(P, T, F, m^i, m^f\right)$ is sound iff:

- *Option to complete*: For any reachable marking m, it is possible to reach marking m^f.
- *Proper completion*: The only reachable marking that contains a token in the final place o is the marking m^f.
- *Absence of dead transitions*: There are no dead transitions, i.e. it does not exist a transition that is not enabled at any reachable marking.

2.3 Explanations of ML Models

In Machine and Deep Learning, the highest accuracy is often achieved by using complex models that are difficult to interpret and, therefore, are often referred to as black box models. In a general sense, a classifier or regressor can be abstracted as a function ψ that returns the estimated value of the target variable, defined over a domain Y. The ψ's domain is defined over the Cartesian product over the domains $X_1, \ldots, X_n$ of the independent features $f_1, \ldots, f_n$, namely $\psi : X_1 \times \ldots \times X_n \to Y$.

Several approaches have been introduced to address the problem of the accuracy-interpretability trade-off. One of the most widespread is based on computing Shapley values, which can be efficiently computed via the method SHAP (SHapley Additive exPlanations) [12]. Shapley values are computed for each feature of each potential element $\vec{x} = (x_1, \ldots, x_n) \in X_1 \times \ldots \times X_n$: given a feature f_i, the Shapley value $\phi_i^{\vec{x}}$ intuitively expresses how much the feature value x_i contributes to the model prediction when the input is $\vec{x}$. The contribution is computed as the average prediction difference with respect to a certain base value, when feature f_i takes on value x_i and each other feature is either discarded or retained. The base value is computed as the average of the predictions when a given multiset $\mathcal{X} \in \mathbb{B}(X_1 \times \cdots \times X_m)$ is used as input (usually the model's training set).[1] Space limitation prevents us from providing a full introduction of the SHAP theory. Interested readers can refer to [12] for further details.

Shapley values are computed for individual input vectors and individual features. However, we need to obtain a general contribution value for each individual features, fairly independent of the specific input. We thus introduce the following explanation function, used thereafter in the paper, that returns the average of the Shapley values when a feature assumes a certain value.

Definition 5 (Explanation function). *Let $\mathcal{F} = \{f_1, \ldots, f_m\}$ be the set of all the features, with X_j be the domain of f_j. Let $\psi : X_1 \times \cdots \times X_m \to Y$ be a regressor or a classificator model. Let $\mathcal{X} \in \mathbb{B}(X_1 \times \cdots \times X_m)$ be a multiset of elements in $X_1 \times \cdots \times X_m$. Let us consider any $f_i \in \mathcal{F}$. Given a value $x \in X_i$, we define*

$$\phi_{f_i}(x, \mathcal{X}) = \underset{\substack{\vec{x}=(x_1,\ldots,x_m)\in\mathcal{X} \\ x_i = x}}{avg} \left(\phi_i^{\vec{x}}\right) \tag{1}$$

[1] $\mathbb{B}(X)$ indicates the set of all the multisets with element in X.

as the average of the Shapley values ϕ_i computed over the set of input values $\mathcal{X}$, when the feature f_i assumes values $x \in X_i$.

Intuitively, $\phi_{f_i}(x, \mathcal{X})$ expresses the expected impact of feature f_i taking on value x, i.e. how much the fact $f_i = x$ influences the prediction.

3 Framework

The overall idea of this paper is to run simulations over a process model, to obtain a simulated log and then compare it to the real one using a Machine Learning discriminator model. Then, SHAP is used to compute the discriminating features, pinpoint the differences between the two event logs, and repair the process model in an automated iterative way.

The framework can be summarized by the schema illustrated in Figure 1 and can be divided into four steps. In Step 1, the initial process model is used to generate a simulated event log. In Step 2, the traces of the real and simulated event logs are encoded yielding to a dataset used by a Machine Learning discriminator model to detect the inaccuracies of the process model. Step 3 focuses on the identification of the discriminating features, measuring the SHAP feature importance, which is then used in a Step 4 to repair the process model. If the changes are significant, the four steps are repeated; otherwise, the repairing procedure terminates. Sections from 3.1 to 3.4 introduces detail the four steps of the framework, while Section 3.5 introduces an alternative, **greedy approach** to incorporate some process-model changes, which are expected at Step 4. Hereafter, we refer to the original, non-greedy approach, namely exactly as described in Section 3.4, as the **complete approach**.

3.1 Step 1: Process Simulation

The framework takes in input an event log $\mathcal{L}^r_{\mathcal{A}}$ over an activity set $\mathcal{A}$, along with a Petri net $N = \left(P, T, F, \lambda, m^i, m^f\right)$ with $\lambda : T \to A \cup \{\tau\}$ (cf. Section 2.2).

The event log is partitioned temporally into training and validation sets: the first part of the traces is allocated to the training set and the remaining traces to the validation set. The training set is used to train the model and discover the discriminatory features, while the validation to validate the tuning of some hyper-parameters, aiming to avoid underfitting and overfitting of the discriminator model.

Petri nets are converted into Stochastic Labelled Petri nets [7], which assign a firing probability to each transition. Burke et al. provides an algorithm to compute these probabilities, which are in fact based on the frequencies of transition firings [7]. A simulated process instance, and hence a trace, is then obtained by looking at the labelled occurrence sequence of the fired transitions.

The simulation is executed for a number of process instances equal to the number of traces in the original event log $\mathcal{L}^r_{\mathcal{A}}$. The simulated event log $\mathcal{L}^s_{\mathcal{A}}$ is then divided into a train and a validation set, with the same number of traces of the splitting in the real event log. This balanced distribution facilitates a fair comparison and evaluation of the framework performance against real data.

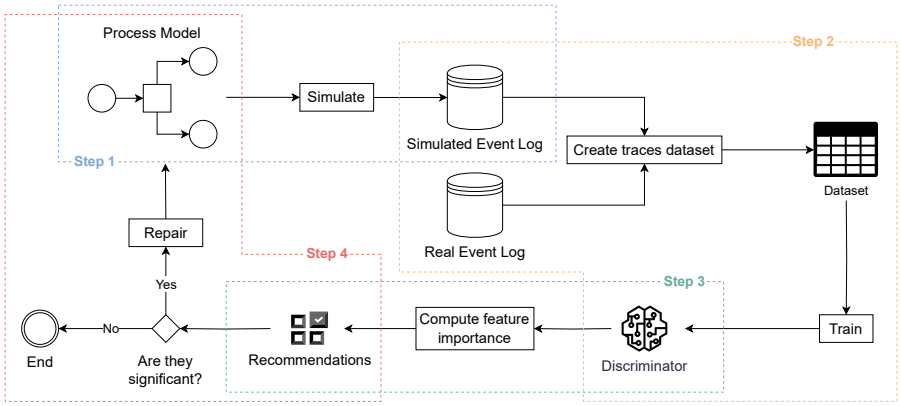


Fig. 1: The framework schema of the proposed approach. Step 1 provides a simulated event log; in Step 2 a dataset of real and simulated traces is created for training a Machine Learning discriminator model; Step 3 regards the computation of the feature importance, providing recommendations for repairing the model; in Step 4 the recommendations are used for automating process model repairs.

3.2 Step 2: Detection of Inaccuracies of Simulation Models

In the second phase, the real and simulated event logs, $\mathcal{L}^r_{\mathcal{A}}$ and $\mathcal{L}^s_{\mathcal{A}}$, are used to discover the log-footprint feature set $\mathcal{F}^{>}_{\mathcal{A}} = \{f_{a>b} \mid (a,b) \in \mathcal{A} \times \mathcal{A}\}$. Note that, since we define a trace to always have the activities $\texttt{start}, \texttt{end} \in \mathcal{A}$, features $f_{\texttt{start}>a}$ and $f_{a>\texttt{end}}$ belong to $\mathcal{F}^{>}_{\mathcal{A}}$, with $a \in \mathcal{A} \setminus \{\texttt{start}, \texttt{end}\}$.

The creation and population of these features are in accodance to the following mapping function:

Definition 6 (Trace-to-Feature Mapping Function). *Let $\mathcal{L}_{\mathcal{A}}$ be an event log. Let $\mathcal{F}^{>}_{\mathcal{A}}$ be the set of footprint features. We define a trace-to-feature mapping function $\rho^{\mathcal{L}_{\mathcal{A}}} : \mathcal{L}_{\mathcal{A}} \to (\mathcal{F}^{>}_{\mathcal{A}} \to \{0,1\})$ s.t. $\rho^{\mathcal{L}_{\mathcal{A}}}(\sigma) = \nu^{\sigma}$ where, given $\sigma = \langle e_1, \ldots, e_m \rangle \in \mathcal{L}_{\mathcal{A}}$, $\forall f_{a>b} \in \mathcal{F}^{>}_{\mathcal{A}}$:*

$$\nu^{\sigma}(f_{a>b}) = \begin{cases} 1 & \text{if } a >_{\sigma} b \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

With these concepts at hand, the training dataset $T_{\mathcal{L}^r_{\mathcal{A}}, \mathcal{L}^s_{\mathcal{A}}}$ of the discriminator model is constructed as follows:[2]

$$T_{\mathcal{L}^r_{\mathcal{A}}, \mathcal{L}^s_{\mathcal{A}}} = \biguplus_{\sigma \in \mathcal{L}^r_{\mathcal{A}}} (\rho^{\mathcal{L}^r_{\mathcal{A}}}(\sigma), 1) \uplus \biguplus_{\sigma \in \mathcal{L}^s_{\mathcal{A}}} (\rho^{\mathcal{L}^s_{\mathcal{A}}}(\sigma), 0) \tag{3}$$

where 1 and 0 indicate that the trace belongs to the real and simulated event logs respectively.

[2] Symbol $\uplus$ indicates the union of multisets.

	$f_{\texttt{start}>a}$	$f_{\texttt{start}>c}$	$f_{a>b}$	$f_{a>c}$	$f_{a>d}$	$f_{b>d}$	$f_{c>d}$	$f_{d>\texttt{end}}$	$f_{c>\texttt{end}}$	*target*
$\langle \texttt{start}, a, b, d, \texttt{end} \rangle$	1	0	1	0	0	1	0	1	0	1
$\langle \texttt{start}, a, c, \texttt{end} \rangle$	1	0	0	1	0	0	0	0	1	1
$\langle \texttt{start}, c, d, \texttt{end} \rangle$	0	1	0	0	0	0	1	1	0	1
$\langle \texttt{start}, a, b, d, \texttt{end} \rangle$	1	0	1	0	0	1	0	1	0	0
$\langle \texttt{start}, a, c, d, \texttt{end} \rangle$	1	0	0	1	0	0	1	1	0	0
$\langle \texttt{start}, a, d, \texttt{end} \rangle$	1	0	0	0	1	0	0	1	0	0

Table 1: Training dataset example: each row represents the encoding of a trace, *target* is equal to 1 if the trace is `real`, 0 if `simulated`. For space reasons, we only show the relevant features, namely those that are not always assigned a value of zero.

Example 1. Given a set of activity labels $\mathcal{A} = \{\texttt{start}, a, b, c, d, \texttt{end}\}$, a real event log $\mathcal{L}^r_{\mathcal{A}} = \langle \langle \texttt{start}, a, b, d, \texttt{end} \rangle, \langle \texttt{start}, a, c, \texttt{end} \rangle, \langle \texttt{start}, c, d, \texttt{end} \rangle \rangle$, and a simulated one $\mathcal{L}^s_{\mathcal{A}} = \langle \langle \texttt{start}, a, b, d, \texttt{end} \rangle, \langle \texttt{start}, a, c, d, \texttt{end} \rangle \langle \texttt{start}, a, d, \texttt{end} \rangle \rangle$, Table 1 describes the returning training dataset: each row represents the encoding of a trace, where the last column indicates if the trace belongs to the real or simulated event log (1 if `real`, 0 otherwise), and the others the correspondent features.

The constructed dataset may have variables with a high correlation, leading to collinearity. When collinearity exists, it can be a challenge to interpret the individual effects of these variables on the target variable. In fact, it becomes difficult to discern whether a feature has a significant impact on the target variable independently or whether its effect is captured by other correlated features in the model. To overcome these issues, a feature selection is performed, considering the correlation between the features: the framework iterates across the features, and for each one, it considers the set of the features with an absolute correlation greater than a given threshold, only retaining one for each of these correlation groups, whose choice can be arbitrarily done.[3] Furthermore, the features that are associated with a variance lower than a fixed threshold are removed. The validation set is then constructed following the same procedure and using the same features selected by the feature selection phase described above.

Then a probabilistic ML model is developed to predict if traces are real or simulated: the output of the model has to be intended as the probability that the input trace is real. The model is trained using the training set $T_{\mathcal{L}^r_{\mathcal{A}}, \mathcal{L}^s_{\mathcal{A}}}$, defined in Equation 3, and the validation set is used to validate the procedure and fine-tune the hyper-parameters of the model.

3.3 Step 3: Identification of the Discriminating Features

Once the model is trained, we create a multiset $\mathcal{X}^{true}$ that is obtained from the training-set instances that the ML model has correctly classified, projected over the independent features (i.e., removing the target variable).[4] This is done in

[3] Usually, suitable thresholds are above 0.9. In our evaluation, we set it to 0.99.

[4] The ML model is in fact probabilistic returning a probability value between 0 and 1. Therefore, an instance with a true value of 1 is considered as correctly classified

order to understand the set of features that enable a correct discrimination. We compute the impact of the features in the output for the samples in $\mathcal{X}^{true}$ to provide explanations of certain classifications (cfr. Section 2.3). For each activity pair $a, b \in \mathcal{A}$:

- If the feature $f_{a>b}$ taking value 1 (or 0) increases (decreases, resp.) the probability for a trace to belong to the real event log, i.e. $\phi_{f_{a>b}}(1, \mathcal{X}^{true}) > 0$ ($\phi_{f_{a>b}}(0, \mathcal{X}^{true}) < 0$), it means that the model does not sufficiently allow for a being followed by b. Therefore, the model needs to enable this.
- If $\phi_{f_{a>b}}(1, \mathcal{X}^{true}) < 0$ (or $\phi_{f_{a>b}}(0, \mathcal{X}^{true}) > 0$), the feature $f_{a>b}$ taking value 1 (0, resp.) impacts the prediction to classify a trace as belonging to the simulated (real, resp.) event log. This means that the real event log does not show that behavior.

In the remainder, $\mathcal{R}$ and $\overline{\mathcal{R}}$ are respectively the set of features related to behavior that needs to be included or not to the model, namely the first or second case in the list above. At this stage, the features that were removed because they were strongly correlated are considered again: any feature exhibiting a strongly positive correlation with another within the set $\mathcal{R}$ is incorporated into $\mathcal{R}$. Similarly, if any feature is negatively correlated to any other in $\overline{\mathcal{R}}$, then it is added to $\mathcal{R}$.

3.4 Step 4: Process Model Repair

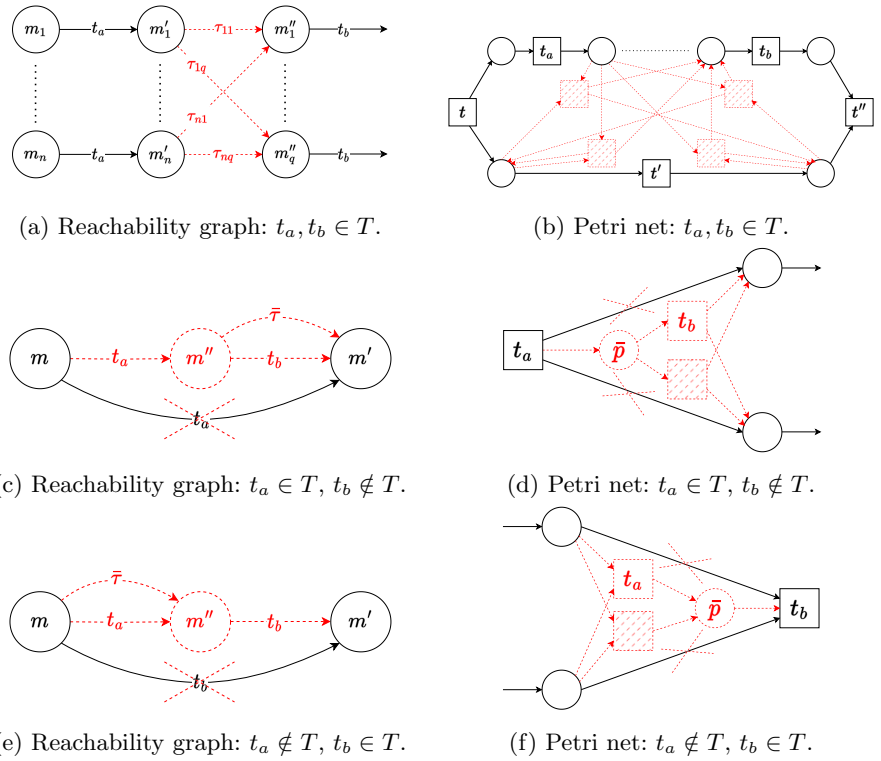
In a final step, the set of features $\mathcal{R}$ is used for repairing the Petri net $N = \left(P, T, F, \lambda, m^i, m^f\right)$, with $\lambda : T \to \mathcal{A} \cup \{\tau\}$, where $\mathcal{A}$ is a set of activity labels, and $G_N = (V, E)$ is the reachability graph. We assume the Petri net N to be sound and safe [2]. This is not very limiting: several process-discovery algorithms, such as Inductive Miner [11] and Split Miner [5], produce Petri nets for which safety holds; also BPMN models with the typical constructs (AND and XOR splits and joins) can be modeled as a safe Petri net. Assuming soundness is also very reasonable: unsound process models cannot be used to enact processes, making them impractical and essentially futile. Repairing models to ensurer soundness is orthogonal to what proposed here, and may be a preliminary step before applying our framework.

To keep the discussion simple, we assume each visible transition to be associated to a different label. However, the alignments can easily be used to lift this up, similarly to what is done, e.g., to compute precision [3]. Hereafter, we indicate with t_l the transition with label $l \in \mathcal{A}$, i.e. $\lambda(t_l) = l$.

For any feature $f_{a>b} \in \mathcal{R}$ we should consider four categories: (i) when there exist net transitions with labels a and b in the original Petri net N, (ii) and (iii) when one of them is missing, (iv) when both are missing. The model is repaired differently for each category:

(i) If $a \not\succ_N b$ and $t_a, t_b \in T$, we alter the Petri net such that the corresponding reachability graph contains one edge (m'_i, τ_{ij}, m''_j), for each marking $m'_1, \ldots, m'_n \in V$ reached after firing transition t_a and for each marking

if the probability value is larger than 0.5. Similarly, an instance with a true value of 0 is correctly classified if the value is smaller than 0.5.



(a) Reachability graph: $t_a, t_b \in T$.

(b) Petri net: $t_a, t_b \in T$.

(c) Reachability graph: $t_a \in T$, $t_b \notin T$.

(d) Petri net: $t_a \in T$, $t_b \notin T$.

(e) Reachability graph: $t_a \notin T$, $t_b \in T$.

(f) Petri net: $t_a \notin T$, $t_b \in T$.

Fig. 2: Petri net repairs. Red and dashed parts denote repairs. We indicate visible transition with label l with t_l, i.e. $\lambda^{-1}(l) = t_l$ if $l \neq \tau$. We assume, without loss of generality, that in category (i) (Figure 2b), there is only one parallelism involving a single transition. For categories (ii) and (iii) (Figures 2d-2f), we consider places to be disjoint. Any differences are negligible.

$m''_1, \ldots, m''_q \in V$ in which t_b is enabled. This is pictorially represented in Figure 2a. The transitions τ_{ij} with $1 \leq i \leq n$ and $1 \leq j \leq q$ are invisible (i.e. $\lambda(\tau_{ij})$ is set to τ) and added to the Petri net. Figure 2b illustrates the resulting Petri net, in a case in which $n = 2$ and $q = 2$, where the inserted invisible transitions are shown in red. The above-mentioned changes in the reachability graph correspond to these specific changes in the Petri net because the Petri net is safe.

(ii) If $t_b \notin T$, we alter the Petri net as follows:
- for each $p \in t^\bullet$, we remove the arc (t_a, p) from F;
- we introduce an visible transition t_b with $\lambda(t_b)$ set to b, an invisible $\overline{\tau}$, and a place $\overline{p}$;
- for each $p \in t^\bullet$, we add the arcs (t_b, p) and $(\overline{\tau}, p)$, along with the arcs $(t_a, \overline{p}), (\overline{p}, t_b), (\overline{p}, \overline{\tau})$.

These changes are visually depicted in Figure 2d, which refers to a case where the outgoing places from t_a are two. Note the category (ii)) is directly elaborated with respect to the Petri net, because the discussion is simpler. Of

course, these changes in the Petri net imply modifications in the reachability graph, as illustrated in Figure 2c.

(iii) If $t_a \notin T$ the procedure is similar to the category (ii) where now we consider the places in ${}^\bullet t_b$. Space limitation prevent us from discussing this case in detail, but Figures 2f and 2e ought to be sufficient for a full understandability.

(iv) If $t_a, t_b \notin T$, then no repair is carried out. However, if $f_{a>b}$ is a discriminatory feature, there is surely a feature $f_{d>a}$ (or $f_{b>d}$, resp.) for some activity d.[5] The model would then be repaired wrt. $f_{d>a}$ (or $f_{b>d}$, resp.), namely the category *(ii)* or *(iii)* would be applied. This would incorporate transition t_a (or t_b) in the model, bringing this repair to category *(ii)* or *(iii)*, which will be incorporated through the subsequent framework iteration.

The proposed repairs guarantee the soundness and safeness of the repaired Petri net if the original one is sound and safe:

Theorem 1. *Let $N = \left(P, T, F, m^i, m^f\right)$ be a sound and safe Petri net system, then the Petri net system N' obtained after applying all repair changes related to every category, namely (i), (ii) or (iii), is sound and safe.*

Proof. The safeness of N' can be easily proven: the only new markings are m_i'' in categories (ii) and (iii), then it is safe by construction.

Regarding soundness, it is sufficient to prove the soundness properties (cf. Section 2.2) for every category.

Option to Complete. For category (i), the reachability graph of the repaired Petri net does not include new states/markings and no arcs are removed: only new arcs are inserted. Therefore, if every marking m of the original reachability graph allowed reaching the final marking m^f, then the same marking m still allows m^f to be reached, at least using the same path. For categories (ii) and (iii), the reachability graph is changed similarly to what depicted in Figures 2c and 2e, respectively. From the newly-introduced marking m'', it is possible to reach marking m', which allowed reaching the final marking in the original Petri net, and so like in the repaired Petri net. Similarly, the original marking m now enables arriving at marking m'', from which, as just mentioned, it is possible to reach the final marking: therefore, it is possible to reach the final marking from m.

Proper completion. Let us define $o \in P$ as the final place, namely $m^f(o) = 1$ and, $m^f(p) = 0$ for all $p \in P \setminus \{o\}$. For category (i), this is straightforward, as no new markings are included and the initial Petri net is sound. For category (ii), we need to prove that $m''(o) = 0$. Let us assume $m''(o) > 0$ by contradiction. Since the repaired Petri nets is safe, there must be $m''(o) = 1$. There are two cases: $m' \neq m^f$ and $m' = m^f$. If $m' \neq m^f$, $m'(o) = 0$ (cf. Figure 2c). Since $m''(o) = 1$, when firing, transitions t_b and $\overline{\tau}$ consume the token in o. However, by construction (cf. Figure 2d), these transitions only consume one token from $\overline{p}$: then, $m''(o) = 0$, contradicting the hypothesis. If $m' = m^f$, $m'(o) = 1$. Since, by hypothesis, $m''(o) = 1$ and transitions t_b and $\overline{\tau}$, when firing, produce one

[5] It might unlikely be the case that d is not present, either. However, this would not invalidate the reasoning, because one repeat it on cascade until one transition is found.

token in o, hence $m'(o) = 2$, which contradicts the safeness properties. Category (iii) is analogous.

Absence of dead transitions. Every transition is present in the reachability graph of the repaired Petri net, which implies that no transition is dead. □

Note that, since the repaired Petri net is kept sound and safe after each change, the different changes can be applied in order. Also, after applying all changes, the entire four phases of our framework can be reiterated. Indeed, during each iteration, the Machine Learning model may uncover new discriminating features, as the training set adapts in response to the evolving process model.

3.5 Extension for a Greedy Repair

Category (i) introduced in Section 3.4 triggers the introduction of several arcs between markings in the reachability graph $G_N = (V, E)$, and consequently brings the introduction of multiple invisible transitions in the original Petri net $N = \big(P, T, F, \lambda, m^i, m^f\big)$. This clearly reduce the model's simplicity and, introducing significant new behavior, may negatively affect the resulting Petri net models.

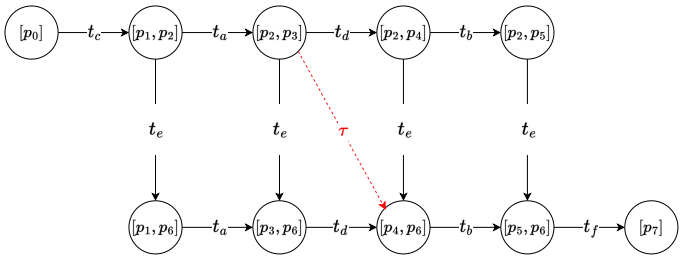
To tackle this challenge, we propose a greedy approach to include relation $a >_N b$ between two transitions $t_a, t_b \in T$. This greedy approach comes from the observation that, given any marking $m_a \in G_N$ reached after firing t_a and any marking $m_b \in G_N$ where t_b is enabled, it is sufficient to add arc (m_a, τ, m_b) to formally incorporate $a >_N b$ into the Petri net. In particular, the approach picks the marking m_a with the shortest path in G_N from the initial marking m_i, along with marking m_b with the short path to the final marking m_f. The approach is motivated by the fact that, by selecting such these two nodes, fewer paths are included in the reachability graph: this leads to a minimization of the behaviour additionally included into the Petri net model, thereby reducing the negative impacts on the precision.

Example 2. Given the Petri net N illustrated in black in Figure 3b, if we want to introduce the relation $a >_N b$ using the greedy approach, only one invisible transition is added to the Petri net model which corresponds to the new arc in the reachability graph highlighted in red in Figure 3a.

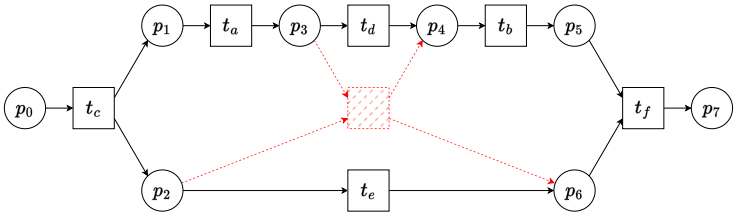
Of course, we acknowledge that this approach only introduces a subset of the potential changes needed to maximize fitness, and, even if we want to introduce a single invisible transition, the choice does not guarantee to be the best to repair the model. This explains why the approach is indeed labelled as greedy. Note that Theorem 1 is still valid for the greedy approach, because we apply a subset of the changes of the complete approach, and each change guarantees soundness and safety, if taken in isolation.

4 Evaluation

In this section, we present the evaluation of our framework, showing our approach is capable of improving the overall accuracy of process models. The experiments conducted focus on computing fitness and precision of the repaired process models at each iteration, and compare them with the state of the art.



(a) The reachabilility graph after repairing.



(b) The repaired Petri net.

Fig. 3: Petri net greedy repairs. Red and dashed parts denote repairs. Markings in the reachability graph are indicated as multisets. We indicate visible transition with label l with t_l, i.e. $\lambda^{-1}(l) = t_l$ if $l \neq \tau$.

4.1 Case Studies

To assess the accuracy and effectiveness of our framework, we evaluated it on five different case studies comprising various business processes:

- Purchase-to-pay (P2P): a synthetic event log describing a purchasing example process.[6]
- BPIC12W: the subprocess for the workflow-relevant activities, i.e. those starting with W, in the BPI Challenge 2012 event data, a log of a loan application process from a Dutch financial institution.[7]
- BPIC17W: the same subprocess as BPIC12W but referring to the 2016 process executions, which are recorded in the BPI Challenge 2017 event data.[8]
- Road Traffic Fines (RTF): process of managing road traffic fines by a local police force in Italy.[9]
- Sepsis: a real life event log obtained from an Enterprise Resource Planning (ERP) system of a regional hospital in The Netherlands.[10]

[6] `https://fluxicon.com/academic/material`
[7] `https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`
[8] `https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b`
[9] `https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5`
[10] `https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460`

The original Petri net models, i.e. before repairs, for the first three case studies were generated by converting the BPMN models publicly available in the repository of the reproducibility package of [8], which were discovered through Split Miner [5]. For all case studies, we also employed the Inductive Miner [11] to mine additional models. To gain insights into the framework's impact on the initial fitness and precision of the models, we experimented with four different noise thresholds $(0.25, 0.5, 0.75, 1)$. It's worth noting that we could obtain the same models using varying noise thresholds. We also performed experiments with a noise threshold equal to 0: with fitness already equal to 1, our greedy and complete technique did not alter the model, and neither did the technique by Fahland et al. [9]. In other words, when the models had fitness equal to 1, all repair techniques returned the original models without any change.

4.2 Experimental Setup

The framework has been implemented in Python: it takes in input an Event Log in XES format, and a Petri net in Petri net Markup Language (PNML) [1].

The event log is temporally split into train, validation, and test sets: with respectively 60%, 20% and 20% of the total traces. The training and the validation logs are used by the framework as described in Section 3, while the test log is employed for evaluation.

In Step 2 (cf. Section 3.2), CatBoost [16] has been used as discriminator model. CatBoost is a Machine Learning method based on gradient boosting over decision trees. SHAP values are computed using the Python's implementation.[11] The validation set/log were also used to perform hyper-parameter optimization during each framework iteration to avoid overfitting and/or underfitting.

The qualities of the obtained model is measured as harmonic means of fitness and precision [3]. The harmonic means aims to quantify the improvement in fitness and/or precision, and their balance. Harmonic means is preferrable over aritmetic means because the former yields more penalization than the latter when fitness or precision is significantly lower than the other (i.e., balancing less). Separately, we also report the simplicity of the models as introduced in [6]. Results have finally been also compared with the state-of-the-art approach by Fahland et al. [9].

4.3 Experimental Results

The results are reported in Table 2, where fitness, precision, their harmonic mean and simplicity are reported for the original model (label O), namely before the repair, our framework using or not greedy repairs (respectively label RG and R and highlighted with a gray background), and the best result by the work of Fahland et al. [9]. Each row indicates a different model. $IM(0.25), \ldots, IM(1)$ denote the original model discovered via Inductive Miner with noise threshold $0.25, \ldots, 1$, while [8] refers to the original "reference" models that are available in the reproducibility package of [8], i.e. obtained by using Split Miner.

[11] Python's documentation for SHAP computation - `https://shap.readthedocs.io`

Case Study	Input Model	Fitness				Precision				H. Mean				Simplicity			
		O	[9]	R	RG	O	[9]	R	RG	O	[9]	R	RG	O	[9]	R	RG
P2P	[8]	0.64	1.00	0.95	0.95	0.99	0.13	0.82	0.85	0.77	0.23	0.88	**0.89**	0.92	0.63	0.78	0.81
	IM(0.25)	0.63	1.00	1.00	1.00	1.00	0.13	0.81	0.81	0.77	0.23	**0.89**	**0.89**	0.96	0.64	0.75	0.75
	IM(0.5,0.75,1)	0.63	1.00	1.00	1.00	1.00	0.13	0.81	0.81	0.78	0.23	**0.89**	**0.89**	1.00	0.64	0.75	0.75
BPIC12W	[8]	0.68	1.00	1.00	1.00	0.91	0.38	0.49	0.49	**0.78**	0.55	0.65	0.65	0.82	0.55	0.60	0.62
	IM(0.25)	0.84	1.00	1.00	1.00	0.91	0.42	0.65	0.65	**0.87**	0.60	0.79	0.79	0.71	0.59	0.61	0.61
	IM(0.5,0.75)	0.62	1.00	0.91	0.91	0.65	0.36	0.49	0.49	0.63	0.53	**0.64**	**0.64**	0.78	0.53	0.60	0.60
	IM(1)	0.61	1.00	1.00	1.00	0.55	0.36	0.36	0.36	**0.58**	0.53	0.53	0.53	0.68	0.48	0.55	0.55
BPIC17W	[8]	0.73	1.00	1.00	1.00	1.00	0.54	0.73	0.73	**0.84**	0.70	**0.84**	**0.84**	0.79	0.59	0.64	0.64
	IM(0.25)	0.85	1.00	1.00	0.98	0.74	0.46	0.70	0.79	0.79	0.63	0.82	**0.87**	0.68	0.55	0.48	0.56
	IM(0.5)	0.57	1.00	0.91	0.79	0.71	0.42	0.61	0.79	0.63	0.59	0.73	**0.79**	0.73	0.51	0.35	0.44
	IM(0.75)	0.58	1.00	0.78	0.67	0.63	0.42	0.57	0.67	0.61	0.59	0.66	**0.67**	0.67	0.51	0.32	0.47
	IM(1)	0.46	1.00	0.79	0.78	0.36	0.42	0.50	0.55	0.40	0.59	0.61	**0.65**	0.71	0.49	0.29	0.42
RTF	IM(0.25)	0.96	1.00	0.99	0.99	0.52	0.44	0.54	0.55	0.68	0.61	**0.70**	**0.70**	0.70	0.55	0.36	0.58
	IM(0.5)	0.89	1.00	1.00	0.98	0.67	0.45	0.45	0.70	0.76	0.62	0.62	**0.81**	0.73	0.57	0.17	0.52
	IM(0.75)	0.93	1.00	0.98	0.98	0.76	0.51	0.78	0.78	0.84	0.68	**0.87**	**0.87**	0.77	0.54	0.59	0.66
	IM(1)	0.61	1.00	0.99	0.99	0.75	0.43	0.78	0.75	0.67	0.60	**0.87**	0.86	1.00	0.52	0.49	0.57
Sepsis	IM(0.25)	0.87	1.00	0.93	0.91	0.49	0.20	0.49	0.49	0.63	0.33	**0.64**	**0.64**	0.67	0.56	0.05	0.56
	IM(0.5)	0.58	1.00	0.95	0.76	0.78	0.20	0.30	0.73	0.67	0.34	0.45	**0.74**	0.75	0.51	0.11	0.33
	IM(0.75,1)	0.52	1.00	0.83	0.55	1.00	0.82	0.44	0.78	0.63	0.34	0.57	**0.65**	0.77	0.50	0.11	0.54
Avg.		0.69	1.00	0.95	0.91	0.76	0.38	0.60	0.67	0.70	0.50	0.72	**0.76**	0.78	0.55	0.45	0.58

Table 2: Experimental results for the five case studies, using the models obtained via Fahland et al. [9], and through our complete (R) and greedy approach (RG). Results are compared through fitness, precision, their harmonic mean, and simplicity. Column O illustrates the values for the original models, namely before repairing, which were either mined through process-discovery techniques or present in literature. The best harmonic mean for each model is highlighted in bold.

The reported results are computed on the test event log, not used for repair. Both the complete and the greedy approach revealed themselves to be effective for almost all models employed in our experiments: compared with 16 out of the original 19 models (85%), they both provide models that improve the harmonic means of fitness and precision, while consistently outperforming the method presented by Fahland et al. [9] (except for one case, where the harmonic means is the same). In fact, the improvement with respect to Fahland et al. [9] is unsurprising, since the method in [9] maximizes fitness by incorporating all the behaviour of the event log into the Petri net model, leading to models with perfect fitness but low precision, whereas our method only adds the most discriminating behaviour without penalizing precision, yielding to a balanced process model. Considering the three original models where we do not improve on the harmonic means, they are already characterized by a high means, which does not allow further improvement. In fact, in an attempt to improve, we worsen the quality of those models: this clearly does not invalidate our technique, because one can easily check whether or not the repaired is actually a better model, and can discard the changes when they are counterproductive.

Comparing our complete approach with our greedy one, we observe that the greedy produces repaired models with higher or similar harmonic means of fitness and precision.

Last, we also compared the level of simplicity of the original and repaired models. Clearly, since our approaches incorporate additional behavior, the repaired models are inevitably less simple. However, a comparison of the simplicity of the models generated by Fahland et al. [9] and by our two approaches, we observe that, on average, our greedy approach produces simpler repaired models: the average simplicity is 0.55 for the approach by Fahland et al. while it is 0.58 for our greedy approach. A visual inspection of the models repaired via our greedy approach shows that the models are clearly readable. Space limitations prevent us from showing how models are repaired for the different case studies. All the models can however be inspected in the Python notebook that belongs to the reproducibility package of this paper [1]. In general, it follows that *our greedy approach is able to produce simpler models with a higher harmonics means of fitness and precision, if compared with Fahland et al. [9], namely the only relevant approach that we were put in the condition to test.*

5 Related Works

The work by Fahland et al. in [9] is among the first ones that address the model-repair problem to enhance fitness, but is not the only. Polyvyanyy et al. [14] puts forward methods to define process model repair as an optimization problem, where costs are assigned to various repair actions. However, this work aims to maximizing the fitness, as authors also clearly state, relegating precision as second-class citizen.

Mitsyuak et al. [13] proposes a model-repair approach able to achieve a high level of fitness, but, as authors themselves admit, at a cost of dropping precision. Other works address the model repair and try to improve the model precision [10, 17]. For the technique in [17], the implementation only works when the original model is mined through Inductive Miner and is fully fitting the event log.

In spite of the limitations mentioned above, we still aimed to compare our technique with those by Polyvyanyy et al. [14], Kalenkova et al. [10,17]. However, we did not manage to run the corresponding reference implementations, and we thus restricted our comparison with the implementation by by Fahland et al. [9].

Other existing literature proposes interactive frameworks for process model repair, in which possible differences between the model and the log are displayed to the user who manually repairs the process model [4,15]. These research works do not aim at an automated model repair, which serves a different purpose: repairing the model, using feedback from humans.

Incremental process discovery (see, e.g., [18]) is also related to model repair, but the goal is different as the former does not aim to obtain models that are as close as possible to a reference original process model.

6 Conclusion

This paper proposes a novel model-repair technique that aims to improve an initial process model so as to maximize the harmonics means of fitness and precision. In fact, a higher harmonics means guarantees that fitness and/or precision are improved and, however, their balance is better. In a nutshell, the process

model is run to generate a simulated event log that is then compared with a real event log using a ML discriminator model. Since an accurate process model would generate logs that are indistinguishable from the original event log, the ML discriminator should perform poorly. If it is conversely able to discriminate, SHAP is used to explain and pinpoint the discriminatory model features. These features are now actionable to allow for an automated model repair: this paper illustrates how to use them to this aim.

The implementation and evaluation of our framework on four distinct processes and various models have proved its ability to generally enhance the original process models and better balance the fitness-precision trade-off. Furthermore, we compared our results with the state of the art, and highlighted that our technique can produce models with a better balance of fitness and precision. As a matter of fact, two repairing approaches are proposed: one more complete, one more greedy in deciding how to repair. While both generally outperform the state of the art, the greedy approach is also applicable to larger models. Furthermore, the experiments have shown that greedy approach is better capable to balance fitness and precision, while producing simpler repaired models.

As future work, we aim to repair multi-perspective process models, by extending the set of features used for comparison, and by incorporating those related to other perspective, such as data, time and resources.

References

1. `https://anonymous.4open.science/r/MLProcessModelRepair-5407`
2. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems and Computers (1998)
3. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining and Knowledge Discovery (2012)
4. Armas-Cervantes, A., van Beest, N.R.T.P., La Rosa, M., Dumas, M., García-Bañuelos, L.: Interactive and incremental business process model repair. In: Proceedings of the On the Move to Meaningful Internet Systems. OTM 2017 (2017)
5. 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 (2017)
6. Blum, F.R.: Metrics in process discovery. In: Technical Report TR/DCC-2015-6, Computer Science Department, University of Chile (2015)
7. Burke, A., Leemans, S.J.J., Wynn, M.T.: Stochastic process discovery by weight estimation. In: Process Mining Workshops - ICPM 2020 International Workshops, Proceedings (2021)
8. Camargo, M., Dumas, M., González-Rojas, O.: Discovering generative models from event logs: data-driven simulation vs deep learning. PeerJ Computer Science (2021)
9. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. Information Systems (2015)
10. Kalenkova, A.A., Carmona, J., Polyvyanyy, A., La Rosa, M.: Automated repair of process models using non-local constraints. In: Application and Theory of Petri Nets and Concurrency - 41st International Conference, PETRI NETS 2020, Proceedings (2020)

11. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - A constructive approach. In: Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Proceedings (2013)
12. Lundberg, S.M., Lee, S.: A unified approach to interpreting model predictions. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Proceedings (2017)
13. Mitsyuk, A.A., Lomazova, I.A., Shugurov, I.S., van der Aalst, W.M.P.: Process model repair by detecting unfitting fragments. In: Supplementary Proceedings of the Sixth International Conference on Analysis of Images, Social Networks and Texts (AIST 2017) (2017)
14. Polyvyanyy, A., van der Aalst, W.M.P., ter Hofstede, A.H.M., Wynn, M.T.: Impact-driven process model repair. ACM Transactions on Software Engineering and Methodology (2017)
15. Pourbafrani, M., van der Aalst, W.M.P.: Interactive process improvement using simulation of enriched process trees. In: Service-Oriented Computing - ICSOC 2021 Workshops - AIOps, STRAPS, AI-PA and Satellite Events, Proceedings (2021)
16. Prokhorenkova, L.O., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A.: Catboost: unbiased boosting with categorical features. In: Advances in Neural Information Processing Systems 31: NeurIPS 2018, Proceedings (2018)
17. Rennert, C., van der Aalst, W.M.P.: Improving precision in process trees using subprocess tree logs. In: ICPM 2023 International Workshops, Proceedings (2023)
18. Schuster, D., Föcking, N., van Zelst, S.J., van der Aalst, W.M.P.: Incremental discovery of process models using trace fragments. In: Business Process Management - 21st International Conference, BPM 2023, Proceedings (2023)