Unblocking Inductive Miner
While Preserving Desirable Properties

Tsung-Hao Huang[0000−0002−3011−9999] and Wil M. P. van der Aalst[0000−0002−0955−6940]

Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
{tsunghao.huang, wvdaalst}@pads.rwth-aachen.de

Abstract. Process discovery aims to discover models to explain the behaviors of information systems. The Inductive Miner (IM) discovery algorithm is able to discover process models with desirable properties: free-choiceness and soundness. Moreover, a family of variations makes IM practical for real-life applications. Due to the advantages, IM is regarded as the state of the art and has been implemented in commercial process mining software. However, IM can only discover block-structured process models that tend to have high fitness but low precision. To improve the quality of process models discovered by IM while preserving desirable properties, we propose an approach that applies property-preserving (free-choiceness and soundness) reduction/synthesis rules to iteratively modify the process model. The experimental results show that the models discovered by our approach have a more flexible representation while preserving desirable properties. Moreover, the model quality, as measured by the F1-score, is improved compared to the original models.

Keywords: Process Discovery · Free-choice Net · Synthesis Rules.

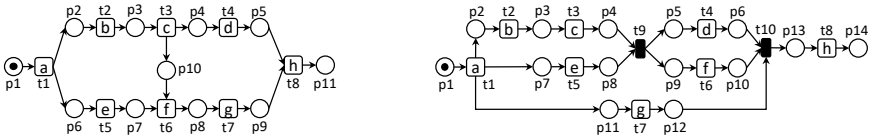
1 Introduction

Process mining provides a wide variety of techniques for stakeholders to gain data-driven insights. To name just a few, process discovery, conformance checking, performance analysis, predictive monitoring, and process enhancement are examples that process mining can offer [1]. Process discovery aims to discover process models that can reflect the behavior of information systems. As a prerequisite for many other techniques, process discovery plays an essential role in a process mining project. Once a process model is discovered from the event log (generated during the process execution in the corresponding information systems), the stakeholders can apply further process mining techniques to generate insights for optimization.

Generally, a discovery algorithm is evaluated by analyzing the process models it discovered. Four dimensions are usually considered: fitness, precision, generalization, and simplicity [1]. Moreover, process models with properties such as free-choiceness and soundness are preferable. On the one hand, a sound process

model ensures the absence of apparent anomalies such as dead components (e.g., transitions that can never be executed) in a model [1]. On the other hand, a free-choice process model has separated constructs for choice and synchronization. Such constructs are naturally embedded in other widely-used and high-level notations such as BPMN (split and join connectors). Consequently, it is straightforward to convert free-choice nets to other notations. Last but not least, an abundance of analysis techniques from theory [12] are available for free-choice models.

The state-of-the-art process discovery algorithm - the Inductive Miner (IM) - can discover process models with the properties mentioned above by exploiting the representation of process trees. By design, the converted Petri net from a process tree is always sound and free-choice. However, a process tree has limited expressive power as the resulting models can only be block-structured, i.e., process models that can be separated into parts with a single entry and exit [20]. As a result, when applying IM to discover models with non-block structures, the quality of the discovered models is often compromised. Specifically, the discovered models usually have high fitness but low precision.



(a) A process model W_{out} with non-block structures. The behaviors cannot be modeled by a process tree without duplicate activity labels.

(b) A model W_{in} discovered by the IMf (with default value 0.2 for the noise threshold) using the log generated by W_{out}.

Fig. 1: An example showing the problem when applying the Inductive Miner to discover a process model with non-block structures. W_{in} allows much more behaviors (low precision) that are not possible in W_{out} as activity g is concurrent to many other activities. At the same time, W_{in} introduces additional constraints that are not in W_{out}, e.g., activity d can only be executed after activity e.

Fig. 1 shows an example that motivates the proposed approach. Fig. 1a is a process model W_{out} with non-block structures. In model W_{out}, there are two concurrent branches after activity a but at the same time, there exists a dependency between them. Using one of the most used IM variants: Inductive Miner - infrequent (IMf) to discover a process model from a log generated by W_{out}, the discovered model would be W_{in} in Fig. 1b. One can see that the behaviors of the two models (W_{out} and W_{in}) are different. Using process trees as the internal representations, no variations of IM can ever discover a model with the same behavior expressed by W_{out} without duplicate activities. Nevertheless, its scalability and guarantees of desirable properties still make it an attractive option for real-life applications.

To improve the quality of the models discovered by the IM while preserving desirable properties, we propose an approach to iteratively modify a model discovered by the IM. Taking a log and the corresponding model (discovered by IM) as input, the approach iteratively modifies the model by applying the reduc-

tion/synthesis rules. Both reduction and synthesis rules are property-preserving (free-choiceness and soundness). Experiments using publicly available real-life event logs show that the quality (w.r.t. F1-score) of the models discovered by IM is indeed improved by our approach. Moreover, the modified models are always sound and free-choice thanks to the property-preserving reduction rules [12,19].

The remainder of the paper is organized as follows. We review the related work in Sec. 2 and define necessary concepts in Sec. 3. Sec. 4 introduces the approach. Sec. 5 presents the experiment and Sec. 6 concludes the paper.

2 Related Work

A comprehensive overview of process discovery approaches can be found in [8,15]. While various process discovery algorithms have been proposed, only a few ensure both soundness and free-choiceness. The Inductive Miner (IM) exploits process trees to guarantee both properties. However, the resulting models are confined to be block-structured. Using process trees to model a process with non-block structures often results in process models with compromised quality. Several approaches [7,9,11] can discover non-block structured models but cannot ensure both properties.

Another group of approaches ensures the desirable properties by applying synthesis rules from free-choice net theory [12]. Dixit et al. [14] were among the first to use synthesis rules for process discovery. The focus was on enabling the interactive setting of process discovery, which requires constant feedback from users with domain knowledge. Nevertheless, the ideal models often cannot be discovered without going back and forth by a combination of reduction and synthesis rules [14]. Furthermore, to recommend to the user the most prominent modifications, the approach needs to evaluate all the possibilities. To address the problems, [19,18] introduces the Synthesis Miner to automate the discovery by introducing predefined patterns using synthesis rules and a search space pruning mechanism.

A closely related field to our proposed approach is process model repair where an existing process model is modified for the purpose of enhancement. Given an existing model and a log containing the behaviors that cannot be replayed by the model, an approach is proposed in [16] to add behaviors to the model locally such that the resulting model can incorporate all the behaviors in the log while staying as similar as possible to the existing model. Instead of fixing all the identified problems (misalignments) as in [16], the approach in [22] prioritizes the changes with the highest impact on fitness. The model repair approaches mentioned so far [16,22] have a tendency toward fitness (being able to replay all the traces) while ignoring other quality dimensions, which often leads to over-generalized models. To avoid the over-generalization pitfall, an interactive repair approach is proposed in [6], where users are expected to provide feedback after viewing the visualization of the mismatches between the event log and the process model. Nevertheless, all the approaches discussed above [16,22,6] do not guarantee sound and free-choice process models.

3 Preliminaries

In this section, we introduce the concepts used throughout this paper. For some set A, $\mathcal{B}(A)$ is the set of all multisets over A. For some multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times element $a \in A$ appears in b. For example, if $A = \{x, y, z\}$, then $b = [x, y^6, z^8] \in \mathcal{B}(A)$ is a multiset consisting of 15 elements. $b(z) = 8$ as z appears eight times in b. $\sigma = \langle a_1, a_2, ..., a_n \rangle \in A^*$ denotes a sequence over A with length $|\sigma| = n$. For $1 \leq i \leq |\sigma|$, $\sigma(i) = a_i$ denotes the i-th element of σ. For instance, $\sigma_s = \langle x, y, x, z \rangle \in A^*$, $|\sigma_s| = 4$, and $\sigma_s(3) = x$. $\langle\rangle$ is the empty sequence. Given two sequences σ and σ', $\sigma \cdot \sigma'$ is the concatenation, e.g., $\langle b \rangle \cdot \langle a, c \rangle = \langle b, a, c \rangle$.

Definition 1 (Sequence Projection). *Let A be a set and $X \subseteq A$ be a subset of A. For $\sigma \in A^*$ and $a \in A$, $\restriction_X \in A^* \to X^*$ is a projection function defined recursively with (1) $\langle\rangle\restriction_X = \langle\rangle$ and (2)*

$$(\langle a \rangle \cdot \sigma)\restriction_X = \begin{cases} \langle a \rangle \cdot \sigma\restriction_X, & \text{if } a \in X \\ \sigma\restriction_X, & \text{otherwise} \end{cases}$$

For example, $\langle a, b, a \rangle\restriction_{\{a,c\}} = \langle a, a \rangle$. Projection can also be applied to multisets of sequences, e.g., $[\langle a, b, c \rangle^6, \langle a, b, b \rangle^6, \langle b, a, c \rangle^2]\restriction_{\{b,c\}} = [\langle b, c \rangle^8, \langle b, b \rangle^6]$.

Definition 2 (Activities, Trace, and Log). *$\mathcal{U}_A$ is the universe of activities. A trace $\sigma \in \mathcal{U}_A^*$ is a sequence of activities. A log is a multiset of traces, i.e., $L \in \mathcal{B}(\mathcal{U}_A^*)$.*

Definition 3 (Petri Net & Labeled Petri Net). *A Petri net is a tuple $N = (P, T, F)$, where P is the set of places, T is the set of transitions, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs. A labeled Petri net $N = (P, T, F, l)$ is a Petri net with a labeling function $l \in T \nrightarrow \mathcal{U}_A$ mapping transitions to activities. For any $x \in P \cup T$, $\bullet x = \{y | (y, x) \in F\}$ denotes the set of input nodes and $x\bullet = \{y | (x, y) \in F\}$ denotes the set of output nodes.*

Note that the labeling function could be partial. If a transition $t \in T$ is not in the domain of l, i.e., $t \notin dom(l)$, it has no label. In such a case, we also write $l(t) = \tau$ to indicate that the transition is silent or invisible. In the labeled Petri net W_{in} of Fig. 1b, transition $t9 \notin dom(l)$ so we say that $t9$ is a silent transition. In this paper, we assume the visible transitions of a labeled Petri net have unique labels, i.e., $\forall_{t_1, t_2 \in T \wedge l(t_1), l(t_2) \in dom(l)} (l(t_1) = l(t_2) \Rightarrow t_1 = t_2)$.

Definition 4 (Free-choice Net). *Let $N = (P, T, F)$ be a Petri Net. N is a free-choice net if for any $t_1, t_2 \in T : \bullet t_1 = \bullet t_2$ or $\bullet t_1 \cap \bullet t_2 = \emptyset$.*

Observe that both nets in Fig. 1 are free-choice nets.

Definition 5 (Path, Elementary Path, Strongly Connected Petri net). *A path of a Petri net $N = (P, T, F)$ is a non-empty sequence of nodes $\rho = \langle x_1, x_2, ..., x_n \rangle$ such that $(x_i, x_{i+1}) \in F$ for $1 \leq i < n$. ρ is an elementary path if $x_i \neq x_j$ for $1 \leq i < j \leq n$. For $X, X' \in P \cup T$, $elemPaths(X, X', N) \subseteq (P \cup T)^*$ is the set of all elementary paths from some $x \in X$ to some $x' \in X'$. N is strongly connected if for any two nodes x and y, there is a path from x to y.*

Definition 6 (Marking). *Let $N = (P, T, F)$ be a Petri Net. A marking $M \in \mathcal{B}(P)$ is a multiset of places. (N, M) is a marked Petri net.*

A transition t is *enabled* in marking M if each of its input places has a token, i.e., $\forall_{p \in \bullet t} M(p) > 0$. Tokens are graphically represented as black dots. An enabled transition can fire. Firing a transition consumes a token from each of its input places and produces a token for each output place. A transition is dead in marking M if no reachable marking enables t.

Definition 7 (Workflow Net (WF-net)). *Let $N = (P, T, F, l)$ be a labeled Petri net, $M_{init}, M_{final} \in \mathcal{B}(P)$ be the initial and final marking respectively. A workflow net (WF-net) is a triplet (N, M_{init}, M_{final}) such that (1) there exists a source place $i \in P : \bullet i = \emptyset$ and a sink place $o \in P : o\bullet = \emptyset$. (2) $M_{init} = [i]$ and $M_{final} = [o]$ (3) the net $N' = (P, T', F', l)$ is strongly connected, where $t' \notin T$, $T' = \{t'\} \cup T$ and $F' = F \cup (o, t') \cup (t', i)$.*

In Fig. 1b, W_{in} is a WF-net with $M_{init} = [p1]$ and $M_{final} = [p14]$. An important property of WF-net is soundness. Three properties [1] need to be held for a WF-net to be sound (1) safeness: places cannot have multiple tokens in any reachable marking (2) option to complete: it is always possible to reach the marking in which only the sink place is marked (3) no dead transitions. Both W_{out} and W_{in} in Fig. 1 fulfill the three properties.

Definition 8 (Reachable Markings & Complete Firing Sequences [2]). *Let $W = (N, M_{init}, M_{final})$ be a WF-net with $N = (P, T, F, l)$ a labeled Petri net. $M[t\rangle M'$ denotes that t is enabled in marking M and the resulting marking M' of firing t is $M' = (M \setminus \bullet t) \cup t\bullet$. Let $\sigma \in T^*$ be a sequence of transitions. $M[\sigma\rangle M'$ denotes that there exists a set of marking $M_1, M_2, ..., M_{n+1}$ such that $M_1 = M, M_{n+1} = M'$, and $M[\sigma(i)\rangle M'$ for $1 \leq i \leq n$, i.e., σ is an enabled firing sequence leading from M to M'. M' is a reachable marking from M if $\exists_{\sigma \in T^*} M[\sigma\rangle M'$. $cfs(W) = \{\sigma \in T^* | M_{init}[\sigma\rangle M_{final}\}$ is the set of complete firing sequences of the WF-net W, i.e., all enabled firing sequence leading from the initial marking M_{init} to the final marking M_{final}.*

For W_{in} in Fig. 1a, $\sigma_1 = \langle t1, t2, t3, t5, t7, t9, t4, t6, t10, t8\rangle \in cfs(W_{in})$ is a complete firing sequence. Firing a transition t in a WF-net is equivalent to executing an activity $l(t)$ if $t \in dom(l)$. Applying the labeling function to a sequence, we get the corresponding trace of the WF-net, e.g., $l(\sigma_1) = \langle a, b, c, e, g, d, f, h\rangle$. Note that, if a transition of a complete firing sequence has no label, it is simply skipped in the corresponding trace.

Definition 9 (Traces of a WF-net [2]). *Let $W = (N, M_{init}, M_{final})$ be a WF-net. $lang(W) = \{l(\sigma) | \sigma \in cfs(W)\}$ is the set of traces possible in W. $act^b(W) = \{\sigma(i) | \sigma \in lang(W) \wedge 1 \leq i \leq |\sigma|\}$ is the set of activities possible in W.*

Definition 10 (Adding Artificial Start & End Activities [2]). *$\blacktriangleright \notin \mathcal{U}_A$ and $\blacksquare \notin \mathcal{U}_A$ are two special activities indicating the start and end of a trace. For any log $L \in \mathcal{B}(\mathcal{U}_A^*)$, $\hat{L} = [\langle\blacktriangleright\rangle \cdot \sigma \cdot \langle\blacksquare\rangle | \sigma \in L]$. For any $S \subseteq \mathcal{U}_A^*$, $\hat{S} = \{\langle\blacktriangleright\rangle \cdot \sigma \cdot \langle\blacksquare\rangle | \sigma \in S\}$.*

Definition 11 (Directly-Follows Relations of a Log and a WF-net [2]). *Let $L \in \mathcal{B}(\mathcal{U}_A^*)$ be a log and $W = (N, M_{init}, M_{final})$ be a WF-net.*

- $act(L) = [a \in \sigma | \sigma \in L]$ *is the multiset of activities in log* L.
- $df(L) = [(\sigma(i), \sigma(i+1)) | \sigma \in \hat{L} \wedge 1 \leq i < |\sigma|]$ *is the multiset of directly-follows relations in the log.*
- $df^b(W) = \{(\sigma(i), \sigma(i+1)) | \sigma \in \hat{S} \wedge 1 \leq i < |\sigma|\}$ *(where $S = lang(W)$) is the set of possible directly-follows relations according to* W.

Note that the special start and end activities have been added (Def. 10) to the traces in $\hat{L}$ and $\hat{S}$. For a log $L = [\langle b, c\rangle^8, \langle b, a\rangle^6]$, the log after adding the start/end activities would be $\hat{L} = [\langle \blacktriangleright, b, c, \blacksquare\rangle^8, \langle \blacktriangleright, b, a, \blacksquare\rangle^6]$ and the multiset of directly-follows relations is $df(L) = [(\blacktriangleright, b)^{14}, (b, c)^8, (c, \blacksquare)^8, (b, a)^6, (a, \blacksquare)^6]$.

4 Approach

In this section, we present the proposed approach. As shown in Fig. 2, the approach takes a log L and a WF-net W discovered by the IM as inputs. Internally, the input WF-net is iteratively modified in order to produce a better model w.r.t. F1-score[1], which is the harmonic mean of the fitness and precision measures.

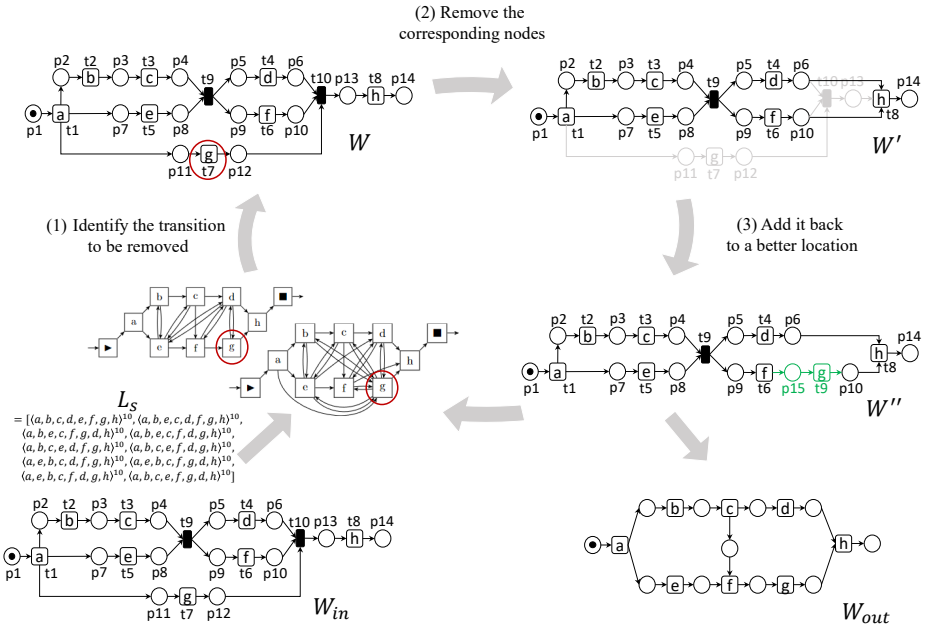


Fig. 2: An example showing a single iteration of our approach.

The general procedure of each iteration is to (1) identify the transition to be removed, (2) remove the corresponding nodes (transitions and possibly also

[1] We use the following formula for the F1-score: $2 \cdot \frac{precision \cdot fitness}{precision + fitness}$.

places) from the WF-net, and (3) add it back to a better location w.r.t. F1-score. In the following subsections, we introduce each step more precisely. We use the following log L_s and the corresponding WF-net W_{in} (in Fig. 1b) discovered by the IM and as the running example.

$$L_s = [\langle a,b,c,d,e,f,g,h\rangle^{10}, \langle a,b,e,c,d,f,g,h\rangle^{10}, \langle a,b,e,c,f,g,d,h\rangle^{10}, \langle a,b,e,c,f,d,g,h\rangle^{10}, \langle a,b,c,e,d,f,g,h\rangle^{10}, \langle a,b,c,e,f,d,g,h\rangle^{10}, \langle a,e,b,c,d,f,g,h\rangle^{10}, \langle a,e,b,c,f,g,d,h\rangle^{10}, \langle a,e,b,c,f,d,g,h\rangle^{10}, \langle a,b,c,e,f,g,d,h\rangle^{10}]$$

4.1 Identification of the transition to be removed

In each iteration, we start by identifying the transition to be removed. We compare the directly-follows relations from the input log L and WF-net W to identify the target transition. The basic idea is to find the activity $a^{\times}$ with the most different directly-follows relations in L and W. Subsequently, the corresponding transition $t^{\times}$ can be identified using the activity label $a^{\times}$.

As noise or infrequent behaviors can pose additional challenges for process discovery algorithms, we filter out the infrequent behaviors in the set of directly-follows relations to make the proposed approach noise-tolerant.

Definition 12 (Filtered Directly-Follows Relations [20]). *Let $L \in \mathcal{B}(\mathcal{U}_A^*)$ be a log and $a \in act(L) \cup \{\blacktriangleright, \blacksquare\}$.*

- *$maxOut(a, L) = max(\{df(L)(a,b) | (a,b) \in df(L)\})$ is the weight of the most frequent directly-follows relations with activity a being the preceding activity.*
- *$dff(L,\omega) = \{(x,y) | ((x,y) \in df(L)) \wedge (df(L)(x,y) \geq maxOut(x,L) \times \omega)\}$ is the set of filtered direclty-follows relations with $0 \leq \omega \leq 1$ as the noise threshold.*

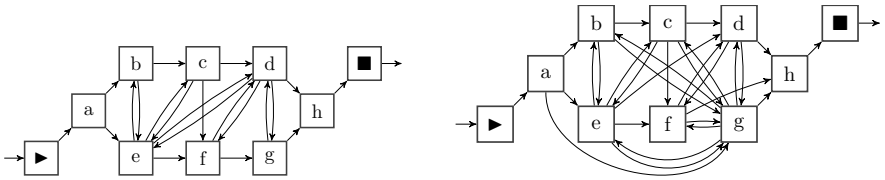
One can think of this as filtering the edges on the corresponding Directly-Follows Graph (DFG). For every activity a, the outgoing edges are filtered out if they occur less than ω times the most frequent outgoing edges of a. The default value for ω is 0.2 [20]. Next, we define the pre- and post-set of activity in a log and a model based on the (filtered) directly-follows relations. Then, the similarity score of an activity is defined based on the differences of its (filtered) directly-follows relations in the log and the model.

Definition 13 (Preset, Postset, and Similarity Score). *Let $L \in \mathcal{B}(\mathcal{U}_A^*)$ be a log, $a \in \mathcal{U}_A^*$ be an activity, and $W = (N, M_{init}, M_{final})$ be a WF-net.*

- *$pre(a, L, \omega) = \{b | (b,a) \in dff(L,\omega)\}$ is the set of activities that activity a directly-follows according to the filtered directly-follows relations $dff(L,\omega)$.*
- *$post(a, L, \omega) = \{b | (a,b) \in dff(L,\omega)\}$. is the set of activities that directly-follow activity a according to the filtered directly-follows relations $dff(L,\omega)$.*
- *$pre^b(a, W) = \{b | (b,a) \in df^b(W)\}$ is the set of activities that activity a directly-follows in the traces possible according to W.*

- $post^b(a, W) = \{b|(a, b) \in df^b(W)\}$ *is the set of activities that directly-follow activity* a *in the traces possible according to* W.
- $sim(a, W, L, \omega) = \frac{1}{2} \times \frac{|pre(a,L,\omega) \cap pre^b(a,W)|}{|pre(a,L,\omega) \cup pre^b(a,W)|} + \frac{1}{2} \times \frac{|post(a,L,\omega) \cap post^b(a,W)|}{|post(a,L,\omega) \cup post^b(a,W)|}$ *is the similarity score of the activity* a *based on the difference of its directly-follows relations in* L *and* W.
- $simMin(W, L, \omega) = \{a \in A | \forall_{b \in A} sim(a, W, L, \omega) \leq sim(b, W, L, \omega)\}$ *(where* $A = act(L) \cap act^b(W)$*) is the set of activities with the lowest similarity score.*

With DFGs, Fig. 3 shows the filtered[2] directly-follows relations of the running example, log L_s (Fig. 3a) and model W_{in} (Fig. 3b). Taking activity h as an example, $pre(h, L_s, 0.2) = \{d, g\}$, $pre^b(h, W_{in}) = \{d, f, g\}$, and $post(h, L_s, 0.2) = post^b(h, W_{in}) = \{\blacksquare\}$. Hence, the similarity score is $sim(h, W_{in}, L_s) = \frac{1}{2} \times \frac{2}{3} + \frac{1}{2} \times 1 = \frac{5}{6}$. The function $simMin(W_{in}, L_s, 0.2) = \{g\}$ calculates the similarity score of all the activities and returns the one(s)[3] with the lowest score, which is activity g in this case. One can also observe that from the two DFGs in Fig. 3, as g is concurrent to all the other activities except for a and h in W_{in}, which results in very different directly-follows relations in the two graphs.



(a) The DFG showing the directly-follows relations of L_s.

(b) The DFG showing the directly-follows relations of W_{in}.

Fig. 3: Two directly-follows graphs (DFGs) showing the directly-follows relations of the running example, log L_s and model W_{in}

Once we get the target activity $a^\times$, the next step is to remove the corresponding transition $t^\times$ from the WF-net. As has been seen in Fig. 2, for our running example, $a^\times = g$ and $t^\times = t7$.

4.2 Removal of the corresponding nodes

Since we are only interested in the WF-nets with desirable properties: soundness and free-choiceness, we need to ensure the removal of the corresponding transition $t^\times$ from the WF-net can keep the properties. Additionally, we would like to make sure that the language of the resulting WF-net W' is the same as the original WF-net W if we only consider the activities that exist in both nets, i.e., $lang(W)\restriction_{act^b(W')} = lang(W')$ (see Def. 1 and 9).

[2] Note that none of the directly-follows relations are filtered out using the default noise threshold for our running example L_s as most of the relations are frequent.

[3] There can be multiple activities with the same lowest similarity score. In such a case, we randomly choose one from the set.

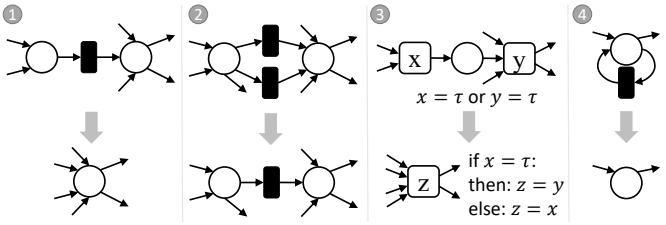


Fig. 4: Behavior preserving reduction rules based on [4,21].

To achieve all the conditions mentioned above, a straightforward and naive approach is to relabel $t^{\times}$ to silent transition τ. However, such a naive solution would potentially leave a lot of silent transitions in the final WF-net. Therefore, after relabeling $t^{\times}$ to be silent, we apply the behavior-preserving reduction rules [4] based on Murata [21] to remove the redundant silent transitions. Fig. 4 shows the reduction rules that are used in this paper.

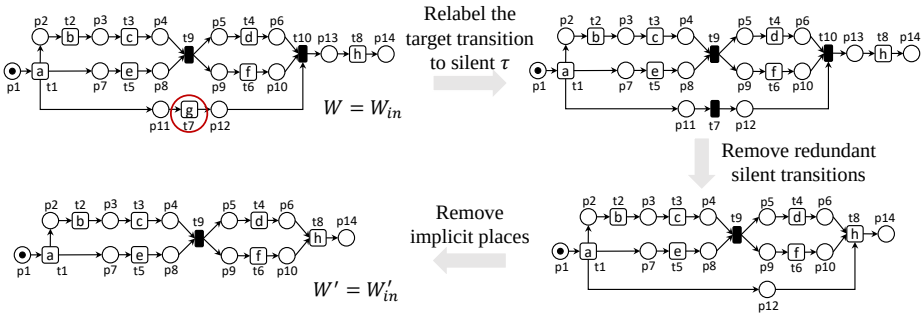


Fig. 5: Applying the reduction step to the running example.

The reduction rules are able to preserve soundness and behaviors but not necessarily free-choiceness. The rule of question is the third one in Fig. 4. For example, let the transition labeled by x be t. Imagine there exists a transition t' with the same input place as t, i.e., $\bullet t' = \bullet t$. After the reduction, the free-choice property can be violated. The reason is that the transition labeled by z and transition t' should have the same set of input places (by Def. 4) but this may not be the case. Thus, we check the free-choice property of the resulting net and only accept the change if it is also a free-choice net.

Removing the silent transitions using the rules specified above can also leave redundant (also called implicit) places in the resulting net. As implicit places have been well-studied and a formal definition is out of the scope, we refer to [10,17] for more details. In general, a place is implicit if removing it does not change the behavior of the net. As shown in Fig. 5, place $p12$ is an implicit place obtained after removing the redundant silent transitions. Implicit places are often not desirable as they can increase the computational complexity of analysis algorithms. Moreover, simplicity and readability are crucial criteria for a process model. Having redundant places in a process model impairs its simplicity. Thus,

the implicit places are removed at the end of the removal step by applying the technique specified in [10]. We denote the resulting net after the removal step as W'. For our running example, $W' = W'_{in}$, as shown in Fig. 5.

4.3 Relocation

In the last step of the iteration, we try to find a better location on the WF-net W' w.r.t. F1-score to add a transition (hereafter denoted as t^+) labeled by $a^\times$. In general, the suitable location to add t^+ should be located between the transitions labeled by the preceding and following (in terms of the causal relations) activities of activity $a^\times$. Thus, we use the causal relationship among activities in the log to identify the preceding and following activities. In the following, we formally define a few log properties to illustrate the idea.

Definition 14 (Log Properties [19]). *Let $L \in \mathcal{B}(\mathcal{U}_A^*)$ be a log and $a, b \in \mathcal{U}_A$ be two activities.*

- $caus(a, b, L) = \begin{cases} \frac{df(L)(a,b)-df(L)(b,a)}{df(L)(a,b)+df(L)(b,a)+1} & \text{if } a \neq b \\ \frac{df(L)(a,b)}{df(L)(a,b)+1} & \text{if } a = b \end{cases}$ *is the strength of causal relation (a, b).*
- $A_\theta^{pre}(a, L) = \{a_{pre} \in \mathcal{U}_A | caus(a_{pre}, a, L) \geq \theta\}$ *is the set of a's preceding activities with a strength of the causal relationship that is at least θ.*
- $A_\theta^{fol}(a, L) = \{a_{fol} \in \mathcal{U}_A | caus(a, a_{fol}, L) \geq \theta\}$ *is the set of a's following activities with a strength of the causal relationship that is at least θ.*

For the running example, we would like to find the preceding and following activities for activity $a^\times = g$ by applying the last two functions in Def. 14. Using the default threshold value for θ (0.9) [19], we get the set of preceding activities as $A_\theta^{pre}(g, L_s) = \{f\}$ and the set of following activities as $A_\theta^{fol}(g, L_s) = \{h\}$. Afterward, we consider every node on the elementary path (Def. 5) between the sets of preceding and following activities as the suitable location. As shown in Fig. 6, the suitable location would be $\{t6, p10, t8\}$.

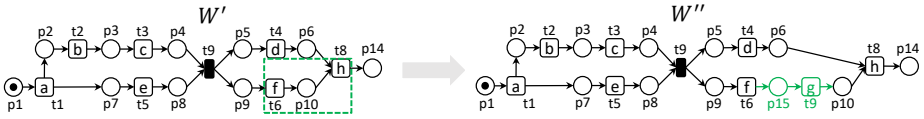


Fig. 6: Adding a transition labeled by activity g back to the WF-net.

Same to the removal step, we also want the resulting net W'' of the relocation step to be free-choice and sound after adding transition t^+ to the suitable location. Accordingly, we apply the patterns defined in [19] to modify the WF-net W' from the removal step. The patterns (including skipping and looping, etc.) are defined based on the synthesis rules introduced in the free-choice net theory [12]. Following the synthesis rules ensures that the two properties can be preserved [13,19]. For the formal definitions of the patterns and rules, we refer to [12,19].

Applying the patterns results in a set of candidates (WF-nets). The candidates are then evaluated based on the alignment-based precision [5] and fitness [3] scores. After that, the candidate with the highest F1-score is selected for the next iteration. As shown in Fig. 6, W'' is the resulting WF-net after adding transition t^+. The loop continues until no further improvements are made w.r.t. the F1-score for three consecutive iterations, which can be set by the users as well.

5 Evaluation

In this section, we evaluate our approach and discuss the experimental results. The approach is implemented[4] in Python using PM4Py[5].

5.1 Experimental Setup

For the experiment, we would like to compare the quality of the WF-nets before and after using our approach. The inputs are an event log L and the corresponding WF-net W discovered by the Inductive Miner. We apply the most widely used IM variation: Inductive Miner-Infrequent (IMf). Two publicly available real-life event logs are used, which are BPI2017[6] and Road Traffic Fine Management[7] (hereafter traffic) respectively. Using the event prefixes, BPI2017 is split into two sub-logs, BPI2017A and BPI2017O. For each log, we apply IMf using six different values (0.0, 0.1, 0.2, 0.3, 0.4, and 0.5) for the noise filter threshold and choose two models with the highest F1-scores. In total, we have six different model-log pairs that will be used as input for our approach.

5.2 Results

Table 1 shows the results of the experiment. The left-hand side of the table records the quality of the input models w.r.t. precision, fitness, and F1-score while the right-hand side of the table records the same for the output models after using our approach.

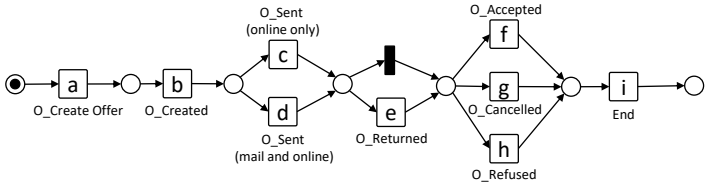
Table 1: Results showing the changes in model quality before/after using our approach.

	before					after			
	IMf-filter	model-id	precision	fitness	F1-score	model-id	precision	fitness	F1-score
BPI2017A	0.2	1	0.936	0.999	0.967	7	0.936	0.999	0.967
	0.4	2	0.999	0.948	0.973	8	0.996	0.960	0.978
BPI2017O	0.2	3	0.907	0.997	0.945	9	0.956	0.997	0.998
	0.5	4	1.000	0.957	0.978	10	1.000	0.989	0.995
traffic	0.2	5	0.555	0.958	0.703	11	0.734	0.961	0.832
	0.4	6	0.752	0.862	0.803	12	0.901	0.865	0.883

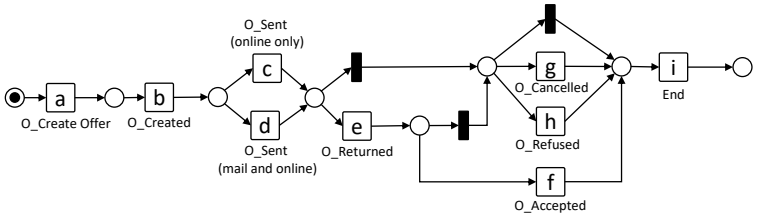
Except for model 1 (whose resulting model (model 7) is essentially the same), the F1-scores of all other models increase. Moreover, almost all of the output

[4] https://git.rwth-aachen.de/tsunghao.huang/unblockIM
[5] https://pm4py.fit.fraunhofer.de/
[6] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[7] https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

models show improvements in both precision and fitness. The only exception appears in models 2 and 8, where there is a trade-off between precision and fitness for a better F1-score. One can observe that the quality of the output models depends on the input model. For the same log, our approach produces a different model depending on the existing model. Such a dependency is expected as the applications of reduction/synthesis rules depend on the existing structure of the original model [13,19].



(a) Before (model-id:3): activity f (*O_Accepted*) does not have to follow activity e (*O_Returned*). However, the behaviors in the log suggest such a restriction.



(b) After (model-id:9): activity f (*O_Accepted*) can only be preceded by activity e (*O_Returned*). Also, activities c, d, and e can now be directly followed by activity i.

Before (model-id: 3)

	a	b	c	d	e	f	g	h	i
a	#	>	#	#	#	#	#	#	#
b	<	#	>	>	#	#	#	#	#
c	#	<	#	#	>	>	>	>	#
d	#	<	#	#	>	>	>	>	#
e	#	#	<	<	#	>	>	>	#
f	#	#	<	<	<	#	#	#	>
g	#	#	<	<	<	#	#	#	>
h	#	#	<	<	<	#	#	#	>
i	#	#	#	#	#	<	<	<	#

Log (BPI2017O)

	a	b	c	d	e	f	g	h	i
a	#	>	#	#	#	#	#	#	#
b	<	#	>	>	#	#	>	>	#
c	#	<	#	#	>	#	>	>	>
d	#	<	#	#	>	#	>	>	>
e	#	#	<	<	#	>	>	>	>
f	#	#	#	#	<	#	#	#	>
g	#	#	<	<	<	#	#	#	>
h	#	#	<	<	<	#	#	#	>
i	#	#	<	<	<	<	<	<	#

After (model-id: 8)

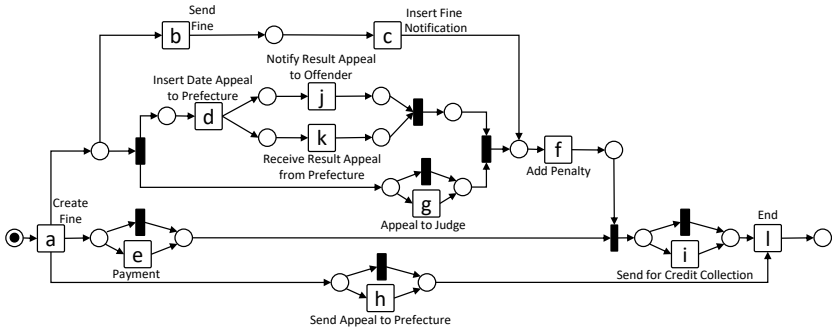
	a	b	c	d	e	f	g	h	i
a	#	>	#	#	#	#	#	#	#
b	<	#	>	>	#	#	#	#	#
c	#	<	#	#	>	#	>	>	>
d	#	<	#	#	>	#	>	>	>
e	#	#	<	<	#	>	>	>	>
f	#	#	#	#	<	#	#	#	>
g	#	#	<	<	<	#	#	#	>
h	#	#	<	<	<	#	#	#	>
i	#	#	<	<	<	<	<	<	#

(c) The footprint matrices of the BPI2017O log and the before/after models (id: 3 and 8). The number of different cells is reduced from 14 to 4. The different four cells stem from the fact that it is possible in the log to skip activity c or d to directly execute g or h after activity b.

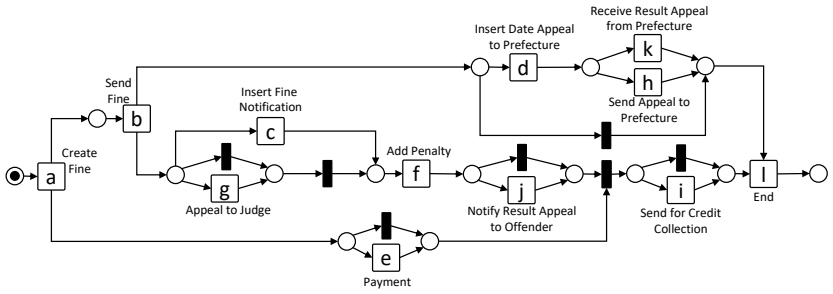
Fig. 7: A comparison of the before/after models for the BPI2017O log.

In addition to the aggregated measurements in Tab. 1, we would like to compare the structures and behaviors of the models before/after using our approach. Fig. 7 and 8 show the comparisons between the before/after models. Additionally, the corresponding footprint matrices[8] are shown in Fig. 7c and 8c. Using the footprint matrix of the log as the ground truth, we highlight the cells of

[8] The cells in the matrix represent the relations between the corresponding two activities. For two activities $x, y \in \mathcal{B}(\mathcal{U}_A^*)$, $x > y$ means that x is directly followed by y but not the other way round. $x\#y$ represents that the two activities never follow each other while $x||y$ means x and y both directly follows each other. For more details and a formal definition of the footprint matrix, we refer to [23].



(a) Before (model-id:6)



(b) After (model-id:12): The modified model has quite some changes except for a few activities such as the start and end activities. First, activity b (*Send Fine*) is placed as the start of two parallel branches. Also, the activities related to "Prefecture" (d,h,k) are now placed closely together as one of the branches after activity b. The self-loop behavior of activity e (*Payment*) is not captured by both models. We assume there is a trade-off between precision and fitness here as allowing self-loop for activity e would probably increase fitness slightly but decrease precision significantly.

Before (model-id: 6)

	a	b	c	d	e	f	g	h	i	j	k	l
a	#	>	#	>	>	>	>	>	#	#	#	#
b	<	#	>	#	||	#	#	||	#	#	#	#
c	#	<	#	#	||	>	#	||	#	#	#	#
d	<	#	#	#	||	#	||	||	#	>	>	#
e	<	||	||	||	#	||	||	||	>	||	||	>
f	<	#	<	#	||	#	<	||	>	<	<	>
g	<	#	#	||	||	>	#	||	#	||	||	#
h	<	||	||	||	||	||	||	#	||	||	||	>
i	#	#	#	#	<	<	#	||	#	#	#	>
j	#	#	#	<	||	>	||	||	#	#	||	#
k	#	#	#	<	||	>	||	||	#	||	#	#
l	#	#	#	#	<	<	#	<	<	#	#	#

Log (traffic)

	a	b	c	d	e	f	g	h	i	j	k	l
a	#	>	#	>	>	#	>	#	#	#	#	#
b	<	#	>	||	||	#	||	||	#	#	#	>
c	#	<	#	||	||	>	>	||	#	#	>	#
d	<	||	||	||	#	||	||	||	>	||	||	>
e	<	||	||	||	||	||	||	||	>	||	||	>
f	#	#	<	||	||	#	||	||	>	||	||	#
g	<	||	<	||	||	||	||	#	||	>	||	||	>
h	#	||	||	||	||	||	||	#	||	||	||	>
i	#	#	#	<	<	<	<	||	#	<	<	>
j	#	#	#	||	||	||	||	||	>	#	||	>
k	#	#	#	<	||	||	||	||	>	||	#	>
l	#	<	#	<	<	#	<	<	<	<	<	#

After (model-id: 12)

	a	b	c	d	e	f	g	h	i	j	k	l
a	#	>	#	#	>	#	#	#	#	#	#	#
b	<	#	>	>	||	>	>	#	#	#	#	#
c	#	<	#	||	||	>	#	||	#	#	||	#
d	#	<	||	#	||	||	||	>	||	||	>	>
e	<	||	||	||	#	||	||	||	>	||	||	>
f	#	<	<	||	||	#	<	||	>	>	||	>
g	#	<	#	||	||	>	#	||	#	#	||	#
h	#	#	||	<	||	||	||	#	||	||	||	>
i	#	#	#	||	<	<	#	||	#	<	||	>
j	#	#	#	||	||	<	#	||	>	#	||	>
k	#	#	||	<	||	||	||	||	||	||	#	>
l	#	#	#	<	<	<	#	<	<	<	<	#

(c) The footprint matrices of the traffic log and the before/after models (id:6 and 12). In total, the number of different cells is reduced from 47 to 39. For most of the activities, the differences either decrease or at least remain except for activities b,g, and h.

Fig. 8: A comparison of the before/after models for the traffic log.

the matrices from the WF-nets in red to indicate differences and in green to represent consensuses.

After using our approach, one can see in Fig. 7c and Fig. 8c that the differences in the footprint matrix are considerably reduced. Also, the resulting models have a more flexible representation (non-block structures) as shown in Fig. 7b and Fig. 8b.

6 Conclusion and Future Work

In this paper, we present an approach that aims to improve the quality of process models discovered by the state-of-the-art Inductive Miner (IM) algorithm while retaining desirable properties, such as free-choiceness and soundness. Our approach iteratively modifies the model by removing problematic transitions (using reduction rules) and adding them back (using synthesis rules) to a better location on the model based on F1-score. By applying rules developed from free-choice net theory [12], we ensure that any modification to the net preserves these desirable properties. Moreover, our approach results in a process model that is not restricted to block-structured, allowing for a more flexible representation and potentially higher quality. We implemented the approach in Python and evaluated it using real-life event logs, with experimental results demonstrating improved quality (measured by F1-score) compared to models discovered by IM alone. Additionally, several resulting process models exhibit non-block structured behaviors, and the directly-follows behaviors of the resulting models are more in line with those from the corresponding log.

There are several possible directions for future work. As the evaluation is limited to two real-life event logs, the restricted experiment post a potential threat to the validity of the approach. Moreover, the experimental results show that it is possible to produce a model without any changes. An open question is whether the approach can improve quality in general. Thus, we plan to conduct a more comprehensive evaluation with more real-life event logs and compare it with existing works. The extended experiment can help to further understand when and how well the approach works. Another direction is to take the similarity between the existing and the modified models into consideration so that the resulting model stays as close as possible to the original one.

Acknowledgements. We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. van der Aalst, W.M.P.: Discovering directly-follows complete Petri nets from event data. In: A Journey from Process Algebra via Timed Automata to Model Learning. Lecture Notes in Computer Science, vol. 13560, pp. 539–558. Springer (2022)
3. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining Knowl. Discov. **2**(2), 182–192 (2012)
4. van der Aalst, W.M.P., Dumas, M., Ouyang, C., Rozinat, A., Verbeek, H.M.W.: Choreography conformance checking: An approach based on BPEL and petri nets. In: The Role of Business Processes in Service Oriented Architectures. Dagstuhl Seminar Proceedings, vol. 06291. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2006)

5. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Measuring precision of modeled behavior. Inf. Syst. E Bus. Manag. **13**(1), 37–67 (2015)
6. Armas-Cervantes, A., van Beest, N.R.T.P., Rosa, M.L., Dumas, M., García-Bañuelos, L.: Interactive and incremental business process model repair. In: OTM Conferences (1). Lecture Notes in Computer Science, vol. 10573, pp. 53–74. Springer (2017)
7. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Bruno, G.: Automated discovery of structured process models from event logs:the discover-and-structure approach. Data Knowl. Eng. **117**, 373–392 (2018)
8. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019)
9. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2019)
10. Berthelot, G.: Transformations and decompositions of nets. In: Advances in Petri Nets. Lecture Notes in Computer Science, vol. 254, pp. 359–376. Springer (1986)
11. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In: BPM. Lecture Notes in Computer Science, vol. 5240, pp. 358–373. Springer (2008)
12. Desel, J., Esparza, J.: Free Choice Petri Nets. No. 40, Cambridge university press (1995)
13. Dixit, P.M.: Interactive Process Mining. Ph.D. thesis, Technische Universiteit Eindhoven (2019)
14. Dixit, P.M., Verbeek, H.M.W., Buijs, J.C.A.M., van der Aalst, W.M.P.: Interactive data-driven process model construction. In: ER 2018. vol. 11157, pp. 251–265. Springer (2018)
15. van Dongen, B.F., de Medeiros, A.K.A., Wen, L.: Process mining: Overview and outlook of Petri net discovery algorithms. Trans. Petri Nets Other Model. Concurr. **2**, 225–242 (2009)
16. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. Inf. Syst. **47**, 220–243 (2015)
17. García-Vallés, F., Colom, J.M.: Implicit places in net systems. In: PNPM. pp. 104–113. IEEE Computer Society (1999)
18. Huang, T., van der Aalst, W.M.P.: Comparing ordering strategies for process discovery using synthesis rules. In: ICSOC Workshops. Lecture Notes in Computer Science, vol. 13821, pp. 40–52. Springer (2022)
19. Huang, T., van der Aalst, W.M.P.: Discovering sound free-choice workflow nets with non-block structures. In: EDOC. Lecture Notes in Computer Science, vol. 13585, pp. 200–216. Springer (2022)
20. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Softw. Syst. Model. **17**(2), 599–631 (2018)
21. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
22. Polyvyanyy, A., van der Aalst, W.M.P., ter Hofstede, A.H.M., Wynn, M.T.: Impact-driven process model repair. ACM Trans. Softw. Eng. Methodol. **25**(4), 28:1–28:60 (2017)
23. Rozinat, A., van der Aalst, W.M.P.: Conformance testing: Measuring the fit and appropriateness of event logs and process models. In: Business Process Management Workshops. vol. 3812, pp. 163–176 (2005)