Fast & Sound: Accelerating Synthesis-Rules-Based Process Discovery

Tsung-Hao Huang[1][0000−0002−3011−9999], Enzo Schneider[2], Marco Pegoraro[1][0000−0002−8997−7517], and Wil M. P. van der Aalst[1][0000−0002−0955−6940]

[1] Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
{tsunghao.huang,pegoraro,wvdaalst}@pads.rwth-aachen.de
http://www.pads.rwth-aachen.de/

[2] RWTH Aachen University, Aachen, Germany
enzo.schneider@rwth-aachen.de

Abstract. Process discovery aims to construct process models describing the observed behaviors of information systems. It is an essential step in process mining projects as most process mining techniques assume a process model as input. While various process discovery algorithms exist, few provide desirable properties: soundness and free-choiceness. By exploiting the free-choice net theory, the recently developed Synthesis Miner not only guarantees the two desirable properties but also enables a more flexible representation (non-block structures) of the discovered process models. The flexibility allows the Synthesis Miner to discover process models with potentially higher quality. Nevertheless, applying the Synthesis Miner remains a challenge due to its lack of scalability. In this paper, we identify the bottleneck and address it by introducing various extensions that utilize the log heuristics and extract the minimal sub-net of the process model. The evaluation using real-life event logs shows that the proposed extensions improve the scalability of the Synthesis Miner by reducing the computation time by 82.85% on average.

Keywords: Process Modeling · Process Mining · Process Discovery · Free-choice Workflow Net · Synthesis Rules

1 Introduction

Process mining is an emerging scientific discipline that bridges the gap between process science and data science. It provides organizations with data-driven techniques to improve operational processes. With the use of process models and event data, process mining techniques help to identify and eliminate inefficiencies in processes.

Process discovery plays an essential role when executing process mining projects because the output of process discovery, a process model, is a prerequisite for many other process mining techniques such as checking conformance, detecting concept drift, predicting performance, etc. The goal of process discovery is to automatically construct a process model from event logs describing the observed behaviors in the corresponding information systems. In general, the quality of a process model can be evaluated by four main criteria — namely fitness, precision, generalization, and simplicity [4]. Additionally, process models with formal guarantees such as soundness and free-choiceness are preferable [4,10]. On the one hand, soundness ensures that the process model does not contain apparent anomalies [4] such as the existence of a dead transition, i.e., a transition that can never be fired. On the other hand, free-choice process models have several benefits. First, a free-choice net has a separate construct for choice and synchronization by definition. Such a construct is also naturally embedded in the widely used process model notation BPMN. Consequently, the property enables easy conversion from free-choice nets to BPMN process models. Moreover, free-choice nets have an abundance of analysis techniques at hand from theory [10].

Despite various algorithms being proposed, few provide the aforementioned two properties. The Inductive Miner (IM) family [15] is one of the few algorithms that ensure such properties. This is achieved by exploiting the representation of process trees, whose converted Petri nets are sound and free-choice by construction. Nevertheless, such a representation is a double-edged sword, as process trees can only represent process models with block structures. Using process trees to represent a process with non-block structures often compromises model quality.

The recently proposed discovery algorithm, the Synthesis Miner [13,14], can discover models with non-block structures while providing the same guarantees by applying the synthesis rules from the free-choice net theory [10]. Adopting an iterative setting, the approach tries to find the best modification (w.r.t. F1-score) to an existing workflow net by generating and evaluating various candidate nets. The generation and evaluation steps require the application of synthesis rules [10] and alignment-based conformance checking [3] respectively. Both operations include expensive computations [13,3]. As a result, adopting the Synthesis Miner remains a challenge due to its scalability problem.

In this paper, we propose various extensions to address the identified bottleneck. The extensions utilize the observation that the modification in each iteration often only affects a subpart of the entire model. The subpart can be extracted and isolated to accelerate the modification. First, log heuristics are exploited to prune the search space even further than the original approach [13,14]. Moreover, the generation and evaluation steps are decomposed into smaller problems by extracting the minimal subnet containing the affected nodes. The extracted subnet can be directly transformed into a sound free-choice WF-net so that the predefined patterns based on synthesis rules can be applied as usual. The experiment using real-life event logs shows that the extensions improve the scalability of the Synthesis Miner by reducing the computation time by 82.85% on average for both generation and evaluation steps.

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

2 Related Work

We refer to [7] for a comprehensive overview of process discovery in general. In this paper, we focus on process discovery algorithms that provide formal guarantees, specifically, soundness and free-choiceness. While various discovery algorithms have been proposed throughout the years, only a handful meet the criteria. The Inductive Miner (IM) family [15] exploits the representation of process trees to provide such guarantees. By definition, every process tree represents a sound and free-choice WF-net. Nevertheless, process trees can only represent processes with block structures, i.e., process models that can be separated into parts with a single entry and exit [15]. As a matter of course, discovery algorithms [16,9] using process trees as internal representation also suffer from the same problem. Various algorithms [8,6] can discover process models with non-block structures but cannot guarantee both free-choice and sound properties.

Another group of algorithms [11,13] utilizes the synthesis rules from free-choice net theory [10] to guarantee both properties while having a more flexible representation. The work in [11] adopts an interactive setting where user inputs are required. However, the discovery process involves various steps of back-and-forth application of synthesis/reduction rules without clear indications. In other words, the users need to have extensive knowledge about the control flow of the process to navigate the discovery. Additionally, although a variation of the work in [11] can recommend the most prominent modifications, it still needs to evaluate all possibilities. The exhaustive search of all the possible modifications is computationally expensive and not feasible in practice.

To address the problems, the Synthesis Miner [13,14] automates the discovery procedure used in [11] by introducing an additional synthesis rule and predefined patterns. The additional rule reduces the need for the back-and-forth steps in [11]. Moreover, using log heuristics, a search space pruning strategy is proposed to locate the most likely position to add the respective transition. The pruning strategy is effective in reducing the computation time [13] as opposed to to evaluating all possibilities. Nevertheless, the Synthesis Miner is still not scalable for process discovery in practice [13,14]. The problem stems from the generation and evaluation steps of the candidates. First, unlikely/undesirable modifications are still included in the set of generated candidates. Including such candidates implies spending unnecessary computation time for evaluation. In addition, there is room for improvement when evaluating the candidates as the modification typically only concerns a subpart of the whole process.

3 Preliminaries

For some set A, $\mathcal{B}(A)$ denotes the set of all multisets over A. For some multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times $a \in A$ appears in b. For example, given a set $A = \{x, y, z\}$, $b = \langle x^5, y^6, z^7 \rangle$ is a multiset over A. $b(y) = 6$ as y appears 6 times in b. $\sigma \in A^*$ denotes that σ is a sequence over some set A. For a sequence $\sigma = \langle a_1, a_2, a_3, ..., a_n \rangle$, $|\sigma| = n$ is the length of σ. For $1 \leq i \leq |\sigma|$,

$\sigma(i) = a_i \in A$ denotes the i-th element of σ. Given two sequences σ and σ', $\sigma \cdot \sigma'$ denotes the concatenation.

Next, we introduce the projection function. Let A be a set and $X \subseteq A$ be a subset of A. For $\sigma \in A^*$ and $a \in A$, we define the projection function $\restriction_X \in A^* \to X^*$ recursively with $\langle\rangle\restriction_X = \langle\rangle$, $(\langle a\rangle \cdot \sigma)\restriction_X = \langle a\rangle \cdot \sigma\restriction_X$ if $a \in X$ and $(\langle a\rangle \cdot \sigma)\restriction_X = \sigma\restriction_X$ otherwise.

Definition 1 (Activities, Traces, and Logs). *Let $\mathcal{A}$ be the universe of activities. A trace $\sigma \in \mathcal{A}^*$ is a sequence of activities. A log $L \in \mathcal{B}(\mathcal{A}^*)$ is a multiset of traces.*

Definition 2 (Petri Net and 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 $l \in T \nrightarrow \mathcal{A}$ mapping transitions to activities. For any $x \in P \cup T$, $\overset{N}{\bullet}x = \{y \mid (y, x) \in F\}$ denotes the set of input nodes (preset) of x and $x\overset{N}{\bullet} = \{y \mid (x, y) \in F\}$ denotes the set of output nodes (postset) of x. The superscript N is dropped if it is clear from the context.*

Note that l can be partial, which means if a transition $t \in T$ is not in the domain of l, it has no label. In such a case, we write $l(t) = \tau$ to denote that the transition is silent or invisible.

Definition 3 (Free-choice Net). *Let $N = (P, T, F)$ be a Petri net. N is a free-choice net if for any $t, t' \in T : \bullet t = \bullet t'$ or $\bullet t \cap \bullet t' = \varnothing$.*

Free-choice nets have separate constructs for choices and synchronizations as any two transitions either have the same preset or don't share any places in their presets.

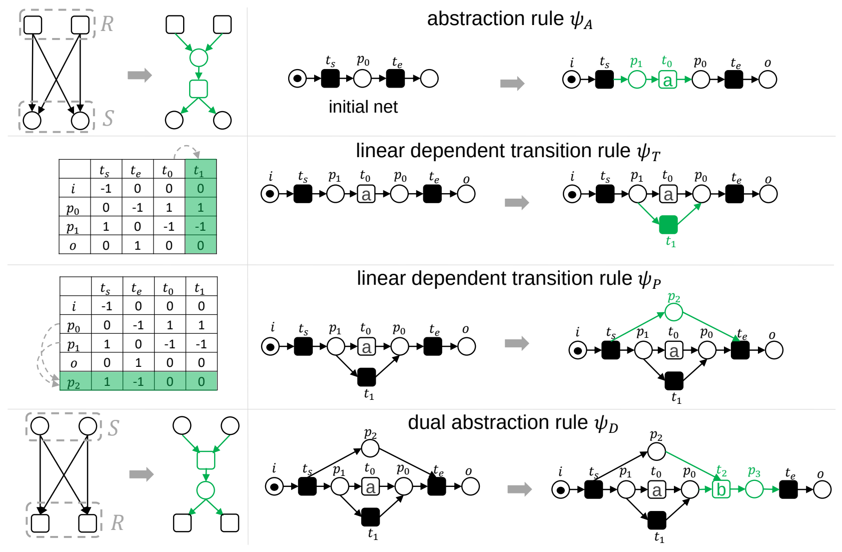
Definition 4 (Path). *A path of a Petri net $N = (P, T, F)$ is a non-empty sequence of nodes $\rho = \langle x_1, x_2, ..., x_n\rangle$ such that $(x_i, x_{i+1}) \in F$ for $1 \leq i < n$.*

Definition 5 (Workflow Net (WF-net)). *Let $N = (P, T, F)$ be a Petri net. N is a workflow net if it has a dedicated source place $i \in P : \bullet i = \emptyset$ and a dedicated sink place $o \in P : o\bullet = \emptyset$. Moreover, every node $x \in P \cup T$ is on some path between i and o.*

The soundness[3] property is defined for WF-nets [1]. A sound WF-net guarantees that (1) a process can always be finished and (2) a process can be properly completed: once a process reaches the final state, it is not possible to fire any transition (3) no inexecutable transitions exist.

Definition 6 (Incidence Matrix [10]). *Let $N = (P, T, F)$ be a Petri net. The incidence matrix $\mathbf{N} : (P \times T) \to \{-1, 0, 1\}$ of N is defined as*

$$\mathbf{N}(p,t) = \begin{cases} 0 & \text{if } ((p,t) \notin F \land (t,p) \notin F) \lor ((p,t) \in F \land (t,p) \in F) \\ -1 & \text{if } (p,t) \in F \land (t,p) \notin F \\ 1 & \text{if } (p,t) \notin F \land (t,p) \in F \end{cases}$$


abstraction rule ψ_A
initial net
linear dependent transition rule ψ_T
linear dependent transition rule ψ_P
dual abstraction rule ψ_D
R
S

	t_s	t_e	t_0	t_1
i	-1	0	0	0
p_0	0	-1	1	1
p_1	1	0	-1	-1
o	0	1	0	0

	t_s	t_e	t_0	t_1
i	-1	0	0	0
p_0	0	-1	1	1
p_1	1	0	-1	-1
o	0	1	0	0
p_2	1	-1	0	0

Fig. 1: Examples of the synthesis rules applications, where the highlighted parts (in green) indicate the newly added components.

Next, we briefly illustrate the synthesis rules introduced in [13,10]. Given a workflow net N, the abstraction rule (ψ_A) allows adding a place p and a transition t between a set of transitions $R \subseteq T$ and a set of places $S \subseteq P$ if they are fully connected, i.e., $(R \times S \subseteq F) \land (R \times S \neq \emptyset)$. The linear transition/place rule (ψ_T/ψ_P) allows adding a transition/place (t/p) if it is linearly dependent on the other transitions/places in the corresponding incidence matrix. Lastly, the dual abstraction rule (ψ_D) allows adding a transition t and a place p between a set of places S and a set of transitions R if $(S \times R \subseteq F) \land (S \times R \neq \emptyset)$. All four rules[4] preserve sound and free-choice properties [10,13]. Fig. 1 (extracted from [12]) shows examples of rules applications. From top to bottom, ψ_A adds p_1 and t_0 with $R = \{t_s\}$ fully connected to $S = \{p_1\}$. ψ_T adds t_1 as it is linearly dependent on t_0. ψ_P adds p_2 as it is a linear combination of p_0 and p_1. ψ_D adds t_2 and p_3 with $S = \{p_0, p_2\}$ fully connected to $R = \{t_e\}$.

4 Approach

In this section, we introduce the extensions that accelerate the computation. As the proposed extensions build on top of the Synthesis Miner [13,14], we briefly discuss the essential steps of the Synthesis Miner and point out limitations that are addressed by the proposed extensions in this paper.

[3] The precise definition of soundness is out of scope, we refer to [1]

[4] The formal definition of the rules is out of scope, we refer to [13,10].

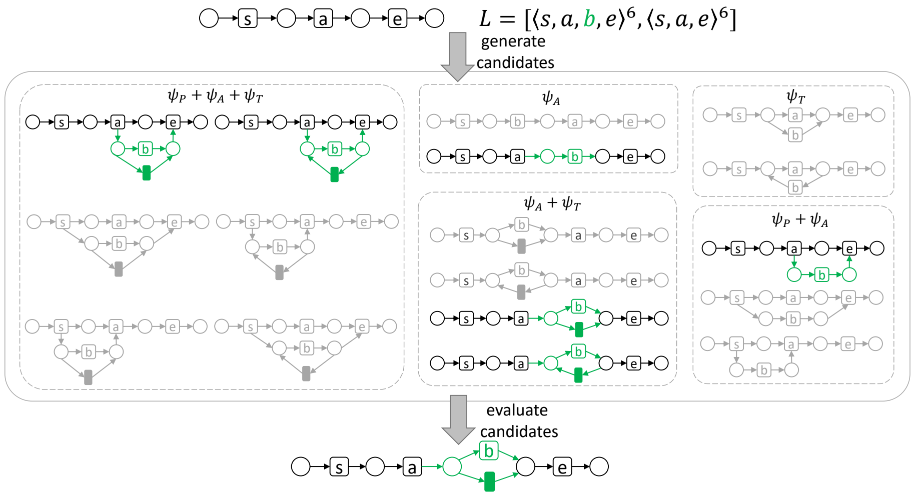


Fig. 2: An example showing the generation and evaluation steps of the Synthesis Miner. First, the candidates are generated using predefined patterns [13] based on synthesis rules, as indicated by the symbols on top of each block. Then, each candidate is evaluated by alignment-based conformance checking to retrieve the F1-score. Note that the figure does not show the complete set of candidates.

4.1 The Synthesis Miner and its limitations

Both variations [13,14] of the Synthesis Miner adopt an interactive approach to modify an existing WF-net that is sound and free-choice. The variation in [13] starts with the *initial net* (as indicated in Fig. 1) containing only the artificial start and end transitions whereas [14] starts from a WF-net discovered by IM.

In every iteration, both variations try to modify the existing net to maximize the model quality (F1-score). The modifications consist of adding new nodes and/or removing existing nodes. Both the removal and addition operations preserve soundness and free-choiceness [14]. Nevertheless, the bottleneck appears when adding new nodes to the existing WF-net. In the background, the addition requires generating and evaluating the candidates. Fig. 2 shows an example of the generation and evaluation steps. In Fig. 2, a transition labeled as b is to be added to the existing net. First, various candidates are generated based on predefined patterns [13] using synthesis rules. Using information from the log, the generation is constrained to connect the new nodes only to the most likely position (existing nodes). The log in Fig. 2 indicates that activity b is preceded by activity a and followed by activity e. To generate the most prominent candidates, the Synthesis Miner connects the new nodes only to the existing nodes on the path between transitions labeled a and e. Any nets with unlikely connections are not considered. For illustration purposes, such nets are still shown in Fig. 2 but grayed out. Then, the candidates are evaluated and the best one is selected for the next iteration based on the F1-score. In this case, the transition labeled

b (with the option to skip) is placed in between transitions labeled a and e. The selected candidate perfectly fits the log (high fitness) and does not allow any unseen behaviors (high precision).

The example in Fig. 2 shows that the pruning strategy used in the existing work [13,14] can reduce the search space to a certain degree. However, the strategy is still unfeasible in real-life scenarios when the event log and the existing net become larger. To discuss the limitations using a concrete example, we consider the following log

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

Limitation 1 (Generation): The main problem in generation is that the search space (number of candidates) grows exponentially with the number of nodes that are considered to be connected to the new node. In [13,14], log heuristics are used to narrow down the search space. To be more precise, we first define a few log properties used to determine the preceding and following activities.

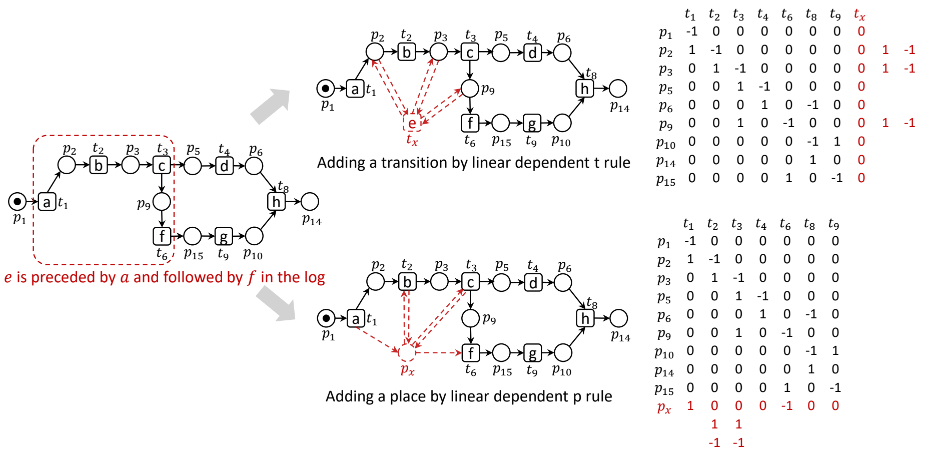
Definition 7 (Log Properties [13]). *Let $L \in \mathcal{B}(\mathcal{A}^*)$ and $a, b \in \mathcal{A}$.*

- $\#(a,L) = \Sigma_{\sigma \in L} |\{1 \le i \le |\sigma| \mid \sigma(i) = a\}|$ *is the times a occurred in L.*
- $\#(a,b,L) = \Sigma_{\sigma \in L} |\{1 \le i < |\sigma| \mid \sigma(i) = a \wedge \sigma(i+1) = b\}|$ *is the number of direct successions from a to b in L.*
- $caus(a,b,L) = \begin{cases} \frac{\#(a,b,L) - \#(b,a,L)}{\#(a,b,L) + \#(b,a,L) + 1} & \text{if } a \neq b \\ \frac{\#(a,b,L)}{\#(a,b,L)+1} & \text{if } a = b \end{cases}$ *is the strength of causal relation (a,b).*
- $A^p_\theta(a,L) = \{a_p \in \mathcal{A} \mid caus(a_p, a, L) \ge \theta\}$ *is the set of a's preceding activities, determined by threshold θ.*
- $A^f_\theta(a,L) = \{a_f \in \mathcal{A} \mid caus(a, a_f, L) \ge \theta\}$ *is the set of a's following activities, determined by threshold θ.*

The following sections assume the use of default value $\theta = 0.9$ [13] to determine the preceding and following activities. Once the sets of preceding and following activities are identified using Def. 7, the corresponding labeled transitions can also be found[5]. Then, every node on the path from the set of preceding transitions to the set of following transitions is considered to be connected to the new node [13].

In Fig. 3, a transition labeled e should be added to the net. Since activity e is preceded by a and followed by f in L_s (Def. 7), the set of nodes on the path is $\{t_1, p_2, t_2, p_3, t_3, p_9, t_6\}$. As an example, using the linear dependent transition rule (ψ_T), we have to check the linear dependency of 3^3=27 vectors as there are three

[5] For the ease of reading, hereafter, we directly refer to the corresponding labeled transitions of the preceding and following activities as the preceding and following transitions.


e is preceded by a and followed by f in the log
Adding a transition by linear dependent t rule
Adding a place by linear dependent p rule
a
b
c
d
e
f
g
h
t_1
t_2
t_3
t_4
t_6
t_8
t_9
t_x
p_1
p_2
p_3
p_5
p_6
p_9
p_{10}
p_{14}
p_{15}
p_x

	t_1	t_2	t_3	t_4	t_6	t_8	t_9	t_x		
p_1	-1	0	0	0	0	0	0	0		
p_2	1	-1	0	0	0	0	0	0	1	-1
p_3	0	1	-1	0	0	0	0	0	1	-1
p_5	0	0	1	-1	0	0	0	0		
p_6	0	0	0	1	0	-1	0	0		
p_9	0	0	1	0	-1	0	0	0	1	-1
p_{10}	0	0	0	0	0	-1	1	0		
p_{14}	0	0	0	0	0	1	0	0		
p_{15}	0	0	0	0	1	0	-1	0		

	t_1	t_2	t_3	t_4	t_6	t_8	t_9
p_1	-1	0	0	0	0	0	0
p_2	1	-1	0	0	0	0	0
p_3	0	1	-1	0	0	0	0
p_5	0	0	1	-1	0	0	0
p_6	0	0	0	1	0	-1	0
p_9	0	0	1	0	-1	0	0
p_{10}	0	0	0	0	0	-1	1
p_{14}	0	0	0	0	0	1	0
p_{15}	0	0	0	0	1	0	-1
p_x	1	0	0	0	-1	0	0
		1	1				
		-1	-1				

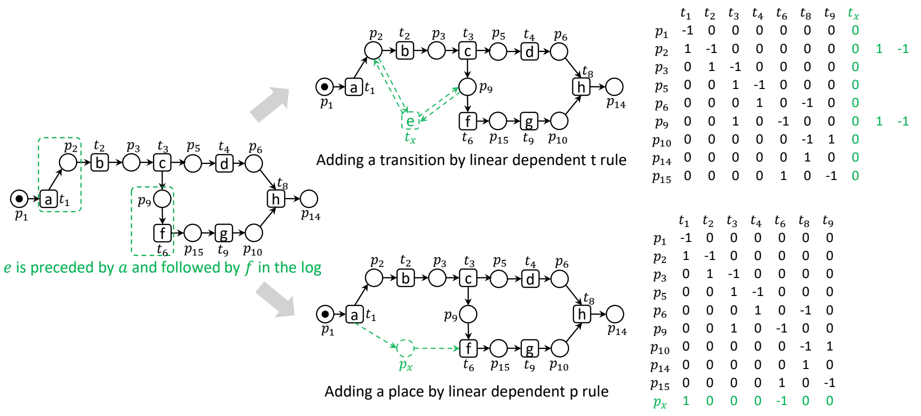
Fig. 3: Adding a transition or a place using the linear dependent rules. The columns and rows outside of the incidence matrix indicate the three possible connections between the new and the existing nodes. Consider the set of nodes $\{t_1, p_2, t_2, p_3, t_3, p_9, t_6\}$ on the path between activity a and f, linear dependency has to be checked on $3^3+3^2=36$ vectors. Note that we only illustrate the basic linear dependent rules in this figure, the best candidate (not shown) in this case combines the linear dependent place rule ψ_P and the abstraction rule ψ_A.

connecting possibilities (corresponding to -1, 1, and 0 in the incidence matrix) for every place in $\{p_2, p_3, p_9\}$. In addition, there are $3^2=9$ vectors to be evaluated when using the linear dependent p rule (ψ_P) to add a place. Furthermore, the predefined patterns are built based on these rules. The example shows that the number of candidates to be evaluated grows exponentially as the number of nodes that are considered for connection. A clear direction for improvement is to further reduce the number of nodes considered for connection.

Limitation 2 (Evaluation) The alignment-based conformance checking is known to be reliable yet computationally expensive [5]. As alignment-based conformance checking is applied to every single candidate for evaluation in [13,14], it introduces a bottleneck in the evaluation phase. Based on observations in [13,14], the change in every iteration often only affects a subpart of the model. Fig. 2 shows an example, where the transition labeled s is not involved in any modification, i.e., no candidates connect new nodes to the transition. The example shows an opportunity to extract and isolate a subnet to speed up the alignment computation.

4.2 Extensions

Extension 1: Log Heuristics As shown in Sec. 4.1, the most likely position (nodes) to place the new node can be identified using heuristics. Although the search space reduction strategy in [13,14] already reduces the computation time significantly, we propose to further reduce the search space by only considering the sets of preceding/following transitions (Def. 7) and their post/pre-sets (Def. 2). Once the sets of preceding/following transitions (T_p/T_f) are identified, the set of nodes to be considered for connection would only be $T_p \cup T_f \cup T_p \bullet \cup \bullet T_f$. Hereafter, we denoted the set of nodes pending for connection as $V_s = T_p \cup T_f \cup T_p \bullet \cup \bullet T_f$. The assumption for the extension is that the other nodes are either in an independent or concurrent relationship with the new node. Thus, there is no need to consider the possibility of adding any connection to the rest of the nodes.



p_1
t_1
p_2
t_2
p_3
t_3
p_5
t_4
p_6
t_8
p_{14}
p_9
t_6
p_{15}
t_9
p_{10}
a
b
c
d
f
g
h
e
t_x
p_x
e is preceded by a and followed by f in the log
Adding a transition by linear dependent t rule
Adding a place by linear dependent p rule

	t_1	t_2	t_3	t_4	t_6	t_8	t_9	t_x		
p_1	-1	0	0	0	0	0	0	0		
p_2	1	-1	0	0	0	0	0	0	1	-1
p_3	0	1	-1	0	0	0	0	0		
p_5	0	0	1	-1	0	0	0	0		
p_6	0	0	0	1	0	-1	0	0		
p_9	0	0	1	0	-1	0	0	0	1	-1
p_{10}	0	0	0	0	0	-1	1	0		
p_{14}	0	0	0	0	0	1	0	0		
p_{15}	0	0	0	0	1	0	-1	0		

	t_1	t_2	t_3	t_4	t_6	t_8	t_9
p_1	-1	0	0	0	0	0	0
p_2	1	-1	0	0	0	0	0
p_3	0	1	-1	0	0	0	0
p_5	0	0	1	-1	0	0	0
p_6	0	0	0	1	0	-1	0
p_9	0	0	1	0	-1	0	0
p_{10}	0	0	0	0	0	-1	1
p_{14}	0	0	0	0	0	1	0
p_{15}	0	0	0	0	1	0	-1
p_x	1	0	0	0	-1	0	0

Fig. 4: Considering only the sets of preceding/following activities and their post/pre-set, the number of vectors pending for linear dependency checkup is reduced from 36 to 3^2+1=10. In this case, the best candidate (not shown in the figure) applies the abstraction rule ψ_A to the net at the bottom to add another place and a transition labeled e in between $R = \{t_1\}$ and $S = \{p_x\}$.

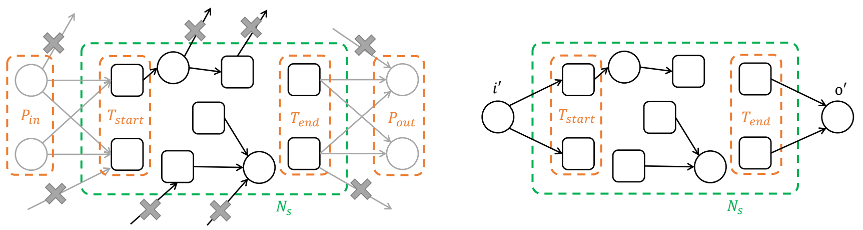
Using the example in Fig. 4 to illustrate the idea, we know that activity e is preceded/followed by activity a/f respectively in log L_s according to Def. 7. The set of nodes considered for connection is then $\{t_1, p_2, p_9, t_6\}$ as opposed to $\{t_1, p_2, t_2, p_3, t_3, p_9, t_6\}$ from the original strategy shown in Fig. 3. As shown in Fig. 4, the number of vectors pending for linear dependency check is reduced from 36 to 10 for the running example. Moreover, the effect of such reduction in turn reduces the number of candidates.

Extension 2: Minimal Subnet Extraction As discussed in Sec. 4.1, not every part of the process model is involved in applying synthesis rules due to the use of log heuristics. In other words, the modification often concerns only a subpart of the model. Motivated by such observation, we propose extracting the subnet containing the set of nodes (V_s) that are most likely to connect to the new nodes according to Extension 1. To be precise, we first define the concept of a subnet in the context of this paper.

Definition 8 (Subnet). *Let $N = (P, T, F)$ be a WF-net that is sound and free-choice. $N_s = (P_s, T_s, F_s)$ is a subnet of N if*

- $P_s \subseteq P, T_s \subseteq T, F_s = F \cap ((P_s \times T_s) \cup (T_s \times P_s))$
- *there exist $P_{in} \subseteq P \backslash P_s, P_{out} \subseteq P \backslash P_s, T_{start} \subseteq T_s, T_{end} \subseteq T_s$ such that*
  - $\forall_{p \in P_{in}} (p\bullet = T_{start})$, *the postset of every place in P_{in} equals to T_{start}.*
  - $\forall_{p \in P_{out}} (\bullet p = T_{end})$, *the preset of every place in P_{out} equals to T_{end}.*
  - $\forall_{t \in T_{start}} (\bullet t = P_{in})$, *the preset of every transition in T_{start} equals to P_{in}.*
  - $\forall_{t \in T_{end}} (t\bullet = P_{out})$, *the postset of every transition in T_{end} equals to P_{out}.*
  - $\forall_{p \in P_s} \big(\overset{N}{\bullet} p \cup p \overset{N}{\bullet} \subseteq T_s\big)$, *all places in P_s only connects to transition in T_s.*
  - $\forall_{t \in T_s \backslash T_{start}} \big(\overset{N}{\bullet} t \subseteq P_s\big)$, *except for T_{start}, the input places of every transition in T_s are in P_s.*
  - $\forall_{t \in T_s \backslash T_{end}} \big(t \overset{N}{\bullet} \subseteq P_s\big)$, *except for T_{end}, the output places of every transition in T_s are in P_s.*

The criteria of a subnet in Def. 9 are visualized in Fig. 5a, where N_s meets all the requirements of a subnet as the only incoming connections are through the start transitions T_{start} and the only outgoing connections are through the end transitions T_{end}. Any other nodes inside N_s have no external connections. A subnet $N_s = (P_s, T_s, F_s)$ can be transformed into a WF-net by adding a source place i' connecting to the set of start transitions T_{start} and a sink place o' connecting from the set of end transitions T_{end}. Fig. 5b shows the transformed WF-net. Note that the transformed WF-net from subnet N_s is sound and free-choice as the whole net N is also a sound free-choice WF-net. Therefore, we can generate candidates based on the transformed WF-net using synthesis rules as usual.



(a) A subnet **(b)** A subnet after transformation

Fig. 5: An example showing the criteria of a subnet as defined in Def. 9 and the transformed WF-net.

Since we are only interested in the smallest subnet containing nodes that might be connected to the new nodes, we define the concept of a minimal subnet.

Definition 9 (Minimal Subnet). *Let $N = (P, T, F)$ be a sound and free-choice WF-net and $N_s = (P_s, T_s, F_s)$ be a subnet of N. Let $V \subseteq P \cup T$. N_s is a minimal subnet for V if*

- *$V \subseteq P_s \cup T_s$ and*
- *there exists no other subnet $N_s^{'} = (P_s^{'}, T_s^{'}, F_s^{'})$ such that*
  - *$V \subseteq P_s^{'} \cup T_s^{'}$*
  - *$(P_s^{'} \cup T_s^{'}) \subset (P_s \cup T_s)$ and $F_s^{'} \subset F_s$.*

The minimal subnet can also be transformed into a sound free-choice WF-net so that the standard synthesis rules in [13] can be used to generate candidates. The benefits of extracting minimal sub-subnets are two-fold.

1. Smaller incidence matrix: applying linear dependent t/p rules requires linear dependency checkup using Gaussian elimination, whose computation complexity is polynomial to the size of the matrix. Therefore, performing Gaussian elimination on a smaller incidence matrix indicates faster computation. The WF-net transformed from the extracted minimal subnet has a smaller incidence matrix. The implication is that any modifications following synthesis rules on the minimal subnet ensure the desirable properties as well.
2. Faster conformance checking: since the modifications are only performed on the extracted subnet, the model quality (w.r.t. F1-score) of the subnet can be used as an indicator for the quality of the whole net. Performing conformance checking on a smaller net and a smaller log also leads to faster computation.

To illustrate the idea, consider the running example as shown in Fig. 6, where the transition labeled as g has to be added to the WF-net. As activity g is preceded by activity f and followed by activity h in log L_s (Def. 7), we know that the set of nodes to be considered for connection would be $V_s = \{t_6, p_{10}, t_8\}$. With the constraint of V_s, the minimal subnet can be extracted as shown in Fig. 6 (highlighted in green). The subnet is then transformed into a WF-net before various candidates are generated and evaluated. Finally, the best candidate is selected and connected back to the original net as shown in the last step in Fig. 6. Moreover, the conformance checking can also be decomposed. Specifically, all candidates derived from modifications on the minimal subnet are evaluated using the projected log $L_s\lceil_{\{d,f,g,h\}} = [\langle d, f, g, h\rangle^{40}, \langle f, g, d, h\rangle^{30}, \langle f, d, g, h\rangle^{30}]$. Then, the best candidate is selected and connected back to the original net. As shown in Fig. 6, the best candidate places the new transition labeled g in between transitions f and h. By removing the source and sink places i' and o', we connect the selected net back to the other part of the original net. Specifically, we add the arcs $(P_{in} \times T_{start}) \cup (T_{end} \times P_{out})$ back.

5 Experimental Evaluation

In this section, we present the experiments conducted to evaluate the efficiency of the proposed extensions. We start by introducing the setup for the experiments before discussing the experimental results.

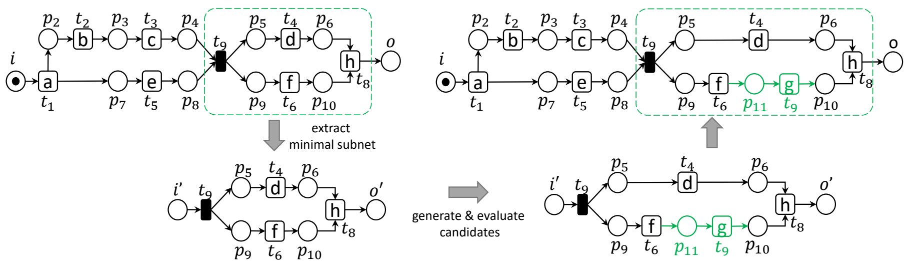


Fig. 6: As activity g is preceded by activity f and followed by activity h in log L_s, the minimal subnet can be extracted. The green dotted line highlights the extracted subnet, where $P_{in} = \{p_4, p_8\}$, $P_{out} = \{o\}$, $T_{start} = \{t_9\}$, $T_{end} = \{t_8\}$

5.1 Experimental Setup

The paper aims to improve the time performance of the synthesis-rules-based process discovery approach introduced in [13,14]. Moreover, the model quality should remain at a similar level. Therefore, comparing the computation time and the model quality with/without the extensions is the experiment's focus. In particular, we would like to focus on the generation and evaluation steps. To achieve this, we use publicly available real-life event logs, which are BPI2017[6] and helpdesk[7]. BPI2017 is split into two sub-logs, BPI2017A and BPI2017O, using the event prefixes.

Since we assume the initial model to be sound and free-choice, we apply Inductive Miner - infrequent (IMf) [15] for this purpose as it guarantees both properties and is scalable at the same time. For each event log, we get two model-log pairs by applying the IMf using two different filter values (0.2 and 0.4). In total, we have 6 model-log pairs as the input for the experiment.

For each model-log pair, we apply the approach in [14] with four different experimental settings, depending on whether the two extensions are turned on/off. For each process model, we remove each labeled transition and add it back. Then, we record the time of generating and evaluating the candidate nets. Lastly, the quality of the resulting models is evaluated and documented. The experiment and the code with extensions can be found in an open repository[8].

5.2 Results and Discussion

Tab. 1 shows the results of the computation time (in seconds) for the generation and evaluation of the candidate nets. Note that the numbers are the average time of adding back every labeled transition in the corresponding process model. As discussed, there are four different settings depending on whether the extensions are applied. The settings are labeled as *e1* (extension 1 is applied), *e2* (extension

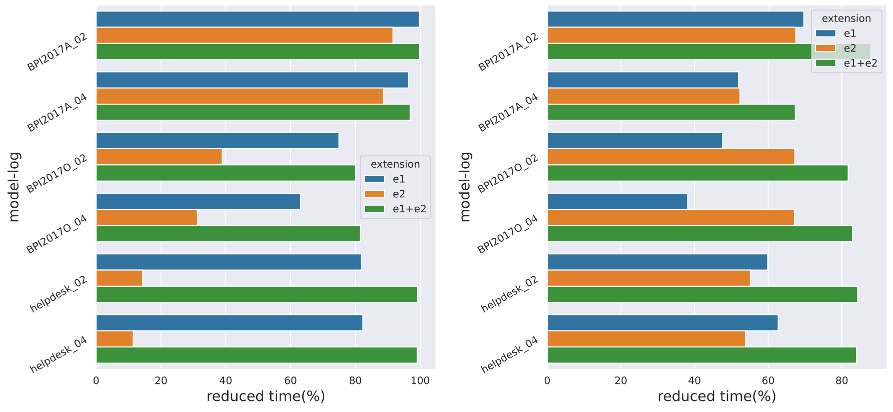
[6] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[7] https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb
[8] https://github.com/denzoned/AccelarateSynthesisMiner

2 is applied), *e1+e2* (both extension 1 and 2 are applied), and *original* (without extensions, correspondent to the original approach in [14]). The model-log pairs are indicated by [log name]_[IMf filter].

Table 1: Computation time (seconds) for generation and evaluation of candidate nets

	generation				evaluation			
model-log	*e1*	*e2*	*e1+e2*	*original*	*e1*	*e2*	*e1+e2*	*original*
BPI2017A_02	0.0990	2.6453	0.0576	31.4863	28.1865	30.2120	11.2979	93.0011
BPI2017A_04	0.0496	0.1566	0.0427	1.3720	17.6582	17.5019	11.9836	36.7301
BPI2017O_02	0.0408	0.0995	0.0325	0.1628	9.2453	5.7846	3.2266	17.6581
BPI2017O_04	0.0559	0.1040	0.0279	0.1514	10.4371	5.5425	2.8860	16.8704
helpdesk_02	0.5753	2.7208	0.0255	3.1770	6.3565	7.1008	2.4854	15.8367
helpdesk_04	0.5131	2.5675	0.0272	2.8994	5.8298	7.2238	2.4993	15.6390

We can see that the computation time is significantly reduced. Fig. 7 shows a clearer indication regarding how much time (%) is reduced compared to the original approach [14]. In general, both extensions can reduce the time for generation and evaluation.



(a) time reduced in generation **(b)** time reduced in evaluation

Fig. 7: Bar charts showing the time reduced (%) compared to the original approach [14]

As shown in Fig. 7a, Extension 1 is more effective in reducing the computation time for generating candidate nets compared to Extension 2. The combination of both extensions reduces the computation time the most, as indicated by the green bars in Fig. 7. On average, both extensions combined reduce the computation time by 82.85% compared to the original approach without any extensions.

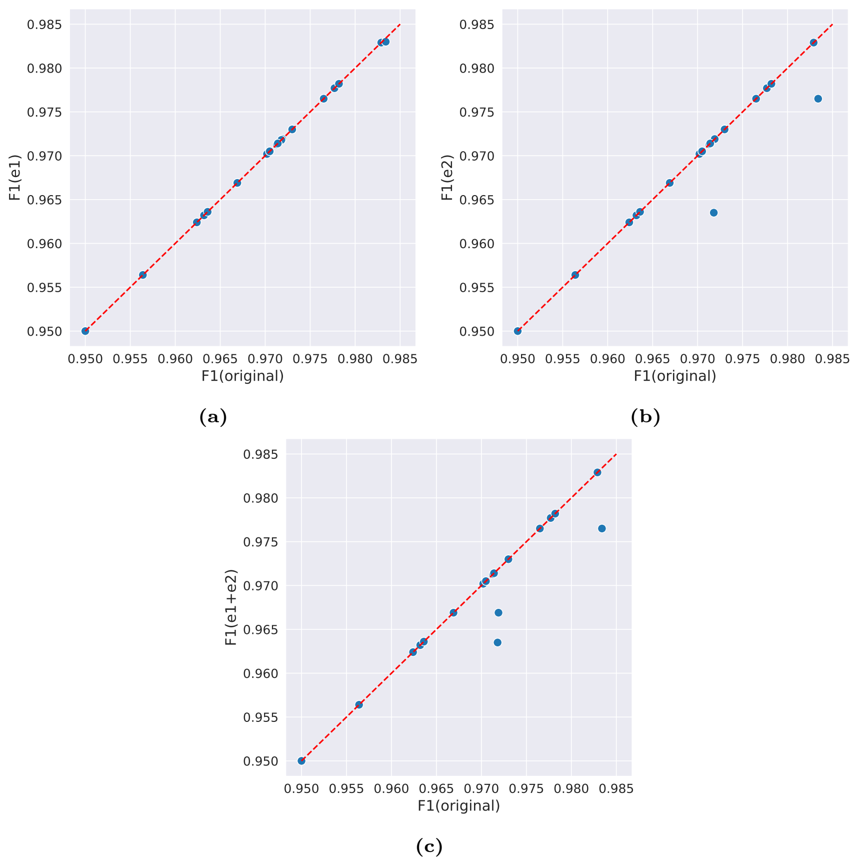


Fig. 8: Scatter plots comparing the quality (F1) of the resulting models using different settings (*e1*, *e2*, *e1*+*e2*). The red dashed lines indicate the ideal situation where the quality of the resulting models with/without applying extensions is the same.

Next, we would like to compare the quality (F1-score) of the resulting process models with/without applying the extensions. Fig. 8 shows the result of the comparison. Each scatter plot consists of 49 data points. For each data point, the x coordinate indicates the F1 score of the resulting model using the original approach [14] whereas the y coordinate shows the same using different extensions (*e1*, *e2*, *e1*+*e2*). Any dot on the red dashed lines indicates the ideal case, where the qualities of the resulting model (when applying the extensions) are the same compared to the original approach. We can see that Setting *e1* always finds the model with the same quality as the original, while settings *e2* and *e1*+*e2* can do the same for most of the models but with a few outliers.

6 Conclusion

The Synthesis Miner algorithm was developed to discover models featuring non-block structures while maintaining desirable guarantees (free-choiceness and soundness) using synthesis rules based on free-choice net theory. Adopting an iterative setting, the approach tries to find the best modification (w.r.t. F1-score) to an existing Workflow net by generating and evaluating various candidate nets. However, the generation and evaluation steps involve the application of synthesis rules and alignment-based conformance checking respectively, presenting scalability challenges.

To address this, our paper proposes extensions leveraging the insight that modifications often impact only a subpart of the model. By isolating these subparts, we aim to speed up the process. Firstly, we use log heuristics to narrow down the search space. Additionally, we break down the generation and evaluation steps into smaller tasks by focusing on a smaller but most relevant component of the process model. Evaluated using real-life event logs, the experiment indicates that the extensions enhance the scalability of the Synthesis Miner by decreasing the computation time by 82.85% on average.

Several directions can be investigated in future works. Firstly, we are interested in techniques that can help to further reduce the search space. One interesting idea could be to apply the evaluation of place fitness [2] to filter out candidate nets containing non-fitting places before evaluation. Also, the scale of the experiment is relatively small, which poses a potential threat to the validity of the extensions. Thus, we plan to conduct a more comprehensive evaluation including more real-life event logs.

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

References

1. van der Aalst, W.M.P.: The application of Petri nets to workflow management. J. Circuits Syst. Comput. **8**(1), 21–66 (1998)
2. van der Aalst, W.M.P.: Discovering the "glue" connecting activities - exploiting monotonicity to learn places faster. In: It's All About Coordination. Lecture Notes in Computer Science, vol. 10865, pp. 1–20. Springer (2018)
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., Carmona, J. (eds.): Process Mining Handbook, Lecture Notes in Business Information Processing, vol. 448. Springer (2022)
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. 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)

7. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019)
8. 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)
9. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: CEC 2012. pp. 1–8. IEEE (2012)
10. Desel, J., Esparza, J.: Free Choice Petri Nets. No. 40, Cambridge university press (1995)
11. Dixit, P.M.: Interactive Process Mining. Ph.D. thesis, Technische Universiteit Eindhoven (2019)
12. 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)
13. 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)
14. Huang, T., van der Aalst, W.M.P.: Unblocking inductive miner - while preserving desirable properties. In: BPMDS/EMMSAD@CAiSE. Lecture Notes in Business Information Processing, vol. 479, pp. 327–342. Springer (2023)
15. Leemans, S.: Robust process mining with guarantees. Ph.D. thesis, Technische Universiteit Eindhoven (2017)
16. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Incremental discovery of hierarchical process models. In: RCIS 2020. vol. 385, pp. 417–433. Springer (2020)