Software and Systems Modeling
https://doi.org/10.1007/s10270-025-01332-y

SPECIAL SECTION PAPER

Flexible and sound: Synthesis Miner

Tsung-Hao Huang[1] · Enzo Schneider[2] · Marco Pegoraro[1] · Wil M. P. van der Aalst[1]

Received: 29 November 2024 / Revised: 18 May 2025 / Accepted: 19 September 2025

Abstract
Process discovery aims to generate models that capture behaviors recorded in event logs from information systems. While many approaches exist, few ensure key properties such as soundness and free-choice. Existing methods that provide these guarantees often rely on block-structured representations, which limit expressiveness. We propose Synthesis Miner, a discovery approach that constructs sound and free-choice workflow nets while supporting non-block structures. Drawing from free-choice net theory, Synthesis Miner incrementally builds or adjusts process models using predefined synthesis patterns and log-guided heuristics to narrow the search space. This ensures correctness by design while improving scalability. We evaluate the approach in two complementary experiments. The first compares the performance of our enhanced Synthesis Miner to its original version, demonstrating substantial reductions in computation time with consistent model quality. The second experiment compares the quality of Synthesis Miner's models against those produced by state-of-the-art discovery algorithms. The results highlight that Synthesis Miner can generate models with competitive, and in some cases superior, quality while ensuring key formal guarantees.

Keywords Process modeling · Process mining · Process discovery · Free-choice workflow net · Synthesis rules

1 Introduction

Process mining is a discipline that bridges process science and data science, offering organizations powerful, data-driven tools to enhance their operational processes. At its core, process mining systematically leverages event data to provide actionable insights into organizational processes. By integrating process models and event logs, it uncovers bottlenecks, deviations, and inefficiencies, anticipates potential compliance or performance issues, and facilitates the automation or streamlining of repetitive tasks. Process mining techniques can be used retrospectively to diagnose problems (e.g., root cause analysis for delays or quality issues) or prospectively to predict outcomes (e.g., remaining processing times or probabilities of failure) [1]. These insights empower organizations to take targeted actions, such as implementing countermeasures to improve compliance or optimize performance.

As the foundation, process discovery creates process models that serve many subsequent techniques, such as conformance checking, concept drift detection, and performance prediction. By automatically generating a process model from event logs, which capture the behaviors recorded in an organization's information systems, process discovery helps organizations gain an understanding of their processes and identify areas for improvement. The discovered model acts as a basis for deeper analysis, enabling organizations to assess the alignment between their actual operations and the intended process design, as well as to predict future behavior or detect deviations.

The goal of process discovery is to automatically extract a process model from an event log containing observed behavior, which is often incomplete and noisy. In general, a process model's quality can be assessed by four main cri-

Communicated by Dominik Bork and Arnon Sturm.

✉ Tsung-Hao Huang
tsunghao.huang@pads.rwth-aachen.de

Enzo Schneider
schneider@isac.rwth-aachen.de

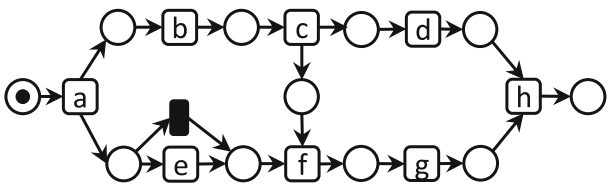
Marco Pegoraro
pegoraro@pads.rwth-aachen.de

Wil M. P. van der Aalst
wvdaalst@pads.rwth-aachen.de

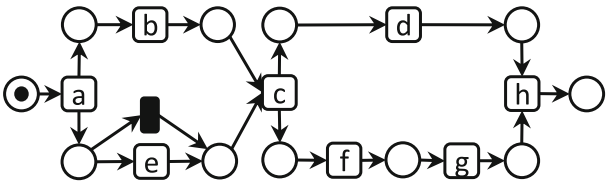
[1] Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany

[2] Institute for Highway Engineering (ISAC), RWTH Aachen University, Aachen, Germany

Published online: 08 November 2025



(a) A model discovered by Synthesis Miner. The models discovered by the Inductive Miner, which internally relies on process trees, are unable to express the same language.



(b) A model discovered by the Inductive Miner using the log generated from the model in (a). The two branches preceding c must be synchronized before d can be executed.

Fig. 1 Examples illustrating the necessity of discovering non-block-structured process models. Note that the trace $\langle a, b, c, d, e, f, g, h\rangle$, which is possible in (**a**), cannot be replayed by (**b**)

teria: fitness, precision, generalization, and simplicity [1]. Additionally, process models that meet formal requirements, such as soundness and free-choice structure, are preferred [1, 2]. Soundness ensures the process model is free from apparent issues, such as dead transitions that can never activate [1]. Free-choice process models offer several advantages, including a clear separation of choice and synchronization, as seen in the popular BPMN notation. This structure enables straightforward conversion from free-choice nets to BPMN models and also allows access to extensive analysis methods rooted in theory [2].

Problem Though there are various process discovery algorithms, few can guarantee both soundness and free-choice properties. The Inductive Miner (IM) family [3] is among the few algorithms that ensure these qualities by representing processes as process trees, which convert to sound, free-choice Petri nets by design. However, process trees have limitations; they only represent block-structured models, which can impact quality when trying to model non-block structures. Using a simple example in Fig. 1a, the process model has a dependency between the two concurrent branches. To execute activity f in the process model, activity c needs to be executed first. At the same time, the execution of activities b, c, and d at the top branch is still concurrent with activity e. Such behaviors cannot be modeled by a block-structured process model (process tree) without duplicated labels. As the Inductive Miner adopts process trees as representation in the background, it can only discover a process model as shown in Fig. 1b for corresponding log generated by the model in Fig. 1a.

Contribution The recently introduced Synthesis Miner [4, 5] addresses this limitation by discovering models with non-block structures while preserving soundness and free-choice guarantees. The approach applies synthesis rules from free-choice net theory in an iterative fashion, modifying a given workflow net to explore different candidate models. However, both the generation of candidates and their evaluation using alignment-based conformance checking can be computationally expensive, especially on larger or more complex logs.

Therefore, this paper proposes two heuristic extensions to improve the scalability of Synthesis Miner without compromising model quality. The first extension applies log heuristics to further narrow the search space compared to the original method. The second focuses on structural decomposition by isolating minimal subnets around affected nodes, breaking the generation and evaluation steps into smaller and more efficient tasks. These improvements preserve the desirable guarantees of the original algorithm while making it more practical for a wider range of discovery tasks.

Results We evaluate the extended Synthesis Miner through two complementary experiments using real-life event logs. The first experiment compares the improved version with the original Synthesis Miner, focusing on scalability. The results show substantial reductions in computation time, particularly when both extensions are combined, while maintaining similar model quality. The second experiment investigates the model quality of Synthesis Miner against established discovery algorithms, including Inductive Miner and Split Miner. Although not all cases benefit equally, the results show that Synthesis Miner can produce models with competitive—and occasionally superior—precision and F1 scores. These findings support the use of Synthesis Miner as a discovery approach that complements existing techniques, especially when model correctness and structural flexibility are required.

Relation to previous work This paper builds upon the work presented in [4–7]. The theoretical foundations of the Synthesis Miner and the details of each step are covered. In this paper, the heuristic extensions (Sects. 4.1 and 4.3) are introduced within the corresponding candidate generation and evaluation steps (Sects. 4.2 and 4.3). Furthermore,

the scale of the experiments (Sect. 5) has been expanded to include 21 model-log pairs, enabling more comprehensive results and a richer discussion on the effectiveness and impact of the proposed extensions. Finally, the extended evaluation allows for a critical reflection on the limitations of Synthesis Miner, alongside its advantages over two state-of-the-art discovery algorithms. The results highlight Synthesis Miner's improvements in scalability and flexibility. The proposed heuristics drastically reduce the computation time, making the approach practical for more complex logs. This enhanced efficiency supports the algorithm's unique ability to discover high-quality, non-block-structured process models. While the capability still comes at a relatively considerable computational cost, it can act as a valuable complementary tool to improve the precision and F1-score of models from other prominent discovery algorithms.

Outline The remainder of the paper is structured as follows. We discuss related work in Sect. 2. Section 3 introduces the necessary background. Section 4 presents the Synthesis Miner and the proposed heuristic extensions. Section 5 describes the experiments. Section 6 discusses the threats to validity and the limitations of the paper. Section 7 concludes the paper with a summary of findings and future research directions.

2 Related works

For a detailed overview of process discovery techniques, we refer the reader to [8]. In this paper, we focus on process discovery algorithms that provide formal guarantees, particularly soundness and free-choiceness. Although numerous discovery algorithms have been developed over the years, only a few meet these criteria.

The Inductive Miner (IM) family [3], which uses process trees as its internal representation, offers such guarantees. By definition, process trees represent sound and free-choice WF-nets. However, process trees can only represent block-structured processes, meaning models that can be divided into components with a single entry and exit point [3]. Naturally, discovery algorithms [9, 10] based on process trees also face this limitation.

While approaches like [11, 12] can discover non-block-structured models, they cannot ensure both soundness and free-choice properties. Structured Miner applies the Heuristics Miner first to discover an unstructured model, then applies block-structuring techniques and heuristics to simplify and remove unsoundness, though it often fails to produce a sound process model with real-life event logs. Split Miner [11] refines the split gateway identification and replaces complex OR-joins with simpler gateways to discover models that are deadlock-free. However, they are not necessarily sound.

Starting from a different motivation, the recently developed Partially Ordered Workflow Language (POWL) [13] addresses the limitations of process trees in modeling hierarchical structures by integrating hierarchical modeling with partial orders. POWL allows for the discovery of models that can represent concurrency and non-block structures, offering a broader expressive power than process trees. The discovered POWL models can be converted into sound WF-nets with non-block structures. However, since its internal representation is still based on process trees, POWL cannot express certain constructs beyond the expressive power of process trees. For instance, arbitrary loop structures remain unsupported, limiting its ability to model complex cyclic behaviors.

Another class of algorithms [4, 14] leverages synthesis rules from free-choice net theory [2], allowing for a more flexible representation while guaranteeing both soundness and free-choice. The approach in [14] adopts an interactive setting, requiring user input throughout the discovery process. However, this method involves repeated back-and-forth applications of synthesis and reduction rules without clear guidance, requiring extensive user knowledge of the process control flow. Although a variation [15] of this approach can recommend prominent modifications, the exhaustive evaluation of all possibilities remains computationally expensive and impractical.

To address these challenges, Synthesis Miner [4, 5] automates the discovery process introduced in [14] by incorporating an additional synthesis rule (dual abstraction rule) and predefined patterns. This additional rule minimizes the need for the repeated back-and-forth steps seen in [14]. Additionally, a log heuristic-based pruning strategy is introduced to narrow down the search space and identify the most probable locations for adding transitions. This pruning strategy significantly reduces computation time [4] compared to evaluating all possible modifications. However, despite these improvements, the Synthesis Miner still struggles with scalability in practice [4, 5]. The main issue arises during the generation and evaluation of candidate modifications. First, the algorithm generates a set of candidates that still includes unlikely or undesirable changes, resulting in unnecessary computation during evaluation. Furthermore, there is potential to optimize the evaluation process, as modifications typically affect only a subpart of the entire process.

3 Preliminaries

We introduce the necessary concepts that are used throughout this paper. For any given set A, $\mathcal{B}(A)$ represents the set of all multisets over A. If $b \in \mathcal{B}(A)$, then $b(a)$ denotes the number of occurrences of $a \in A$ in the multiset b. For example, if we consider the set $A = \{w, x, y, z\}$ and the multiset $b =$

$[w^2, x^4, y^3, z^5]$, we have $b(y) = 3$ because y appears 3 times in b. The notation $\sigma \in A^*$ is used to represent a sequence over some set A. For a sequence $\sigma = \langle a_1, a_2, a_3, ..., a_n \rangle$, the length of the sequence is denoted by $|\sigma| = n$. For each $1 \leq i \leq |\sigma|$, the element $\sigma(i) = a_i \in A$ refers to the i-th element of the sequence. Given two sequences σ and σ', their concatenation is written as $\sigma \cdot \sigma'$.

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 as follows: $\langle\rangle\restriction_X = \langle\rangle$, $(\langle a \rangle \cdot \sigma)\restriction_X = \langle a \rangle \cdot \sigma\restriction_X$ if $a \in X$ and $(\langle a \rangle \cdot \sigma)\restriction_X = \sigma\restriction_X$ otherwise. This allows for the projection of a sequence onto a subset of its elements.

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 do not 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 $1 \leq i < n$.

The foundation of Synthesis Miner lies in the synthesis rules derived from free-choice net theory [2]. Before introducing these rules, it is essential to define a few key concepts. We begin with the definition of the incidence matrix, followed by the concept of linearly dependent nodes.

Definition 5 (Incidence Matrix [2]) 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

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

For a Petri net $N = (P, T, F)$ with an incidence matrix $\mathbf{N}$, we denote the row vector corresponding to $p \in P$ as $\mathbf{N}(p)$ and the column vector corresponding to $t \in T$ as $\mathbf{N}(t)$.

Definition 6 (Linearly Dependent Nodes [2]) Let $N = (P, T, F)$ be a Petri net. $\mathbb{Q}$ is the set of rational numbers. A place p is linearly dependent if there exists a row vector $\vec{v} : P \to \mathbb{Q}$ such that $\vec{v}(p) = 0$ and $\vec{v} \cdot \mathbf{N} = \mathbf{N}(p)$. A transition t is linearly dependent if there exists a column vector $\vec{v} : T \to \mathbb{Q}$ such that $\vec{v}(t) = 0$ and $\vec{v} \cdot \mathbf{N} = \mathbf{N}(t)$.

Definition 7 (Workflow Net (WF-net)) Let $N = (P, T, F, l)$ be a labeled 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[1] property is defined for WF-nets [16]. 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.

Before defining the synthesis rules for WF-nets, it is necessary to introduce the concept of a short-circuited WF-net as the rules work on short-circuited WF-nets in the background.

Definition 8 (Short-circuited WF-net [16]) Let $W = (P, T, F, l)$ be a WF-net and $i, o \in P$ are the source and sink place, respectively. The short-circuited WF-net of W, denoted by $SC(W)$, is constructed by adding a transition $t' \notin T$ and its associated arcs $(o, t'), (t', i)$, i.e., $SC(W)=(P, T \cup \{t'\}, F \cup \{(o, t'), (t', i)\})$, where $t' \notin T$.

Definition 9 (Synthesis Rules [2, 4, 14]) Let W and W' be two free-choice WF-nets, and let $SC(W) = (P, T, F, l)$ and $SC(W') = (P', T', F', l')$ be the corresponding short-circuited WF-nets:

- Linear Dependent Place Rule ψ_P: W' is derived from W using ψ_P, i.e., $(W, W') \in \psi_P$ if $T' = T$, $P' = P \cup \{p\}$ and $p \notin P$ is linear dependent in $SC(W')$, $F' = F \cup \widetilde{F}$ where $\widetilde{F} \subseteq (\{p\} \times T) \cup (T \times \{p\}))$.
- Linear Dependent Transition Rule ψ_T: W' is derived from W using ψ_T, i.e., $(W, W') \in \psi_T$ if $P' = P$, $T' = T \cup \{t\}$ and $t \notin T$ is linear dependent in $SC(W')$

[1] The precise definition of soundness is out of scope, we refer to [16]

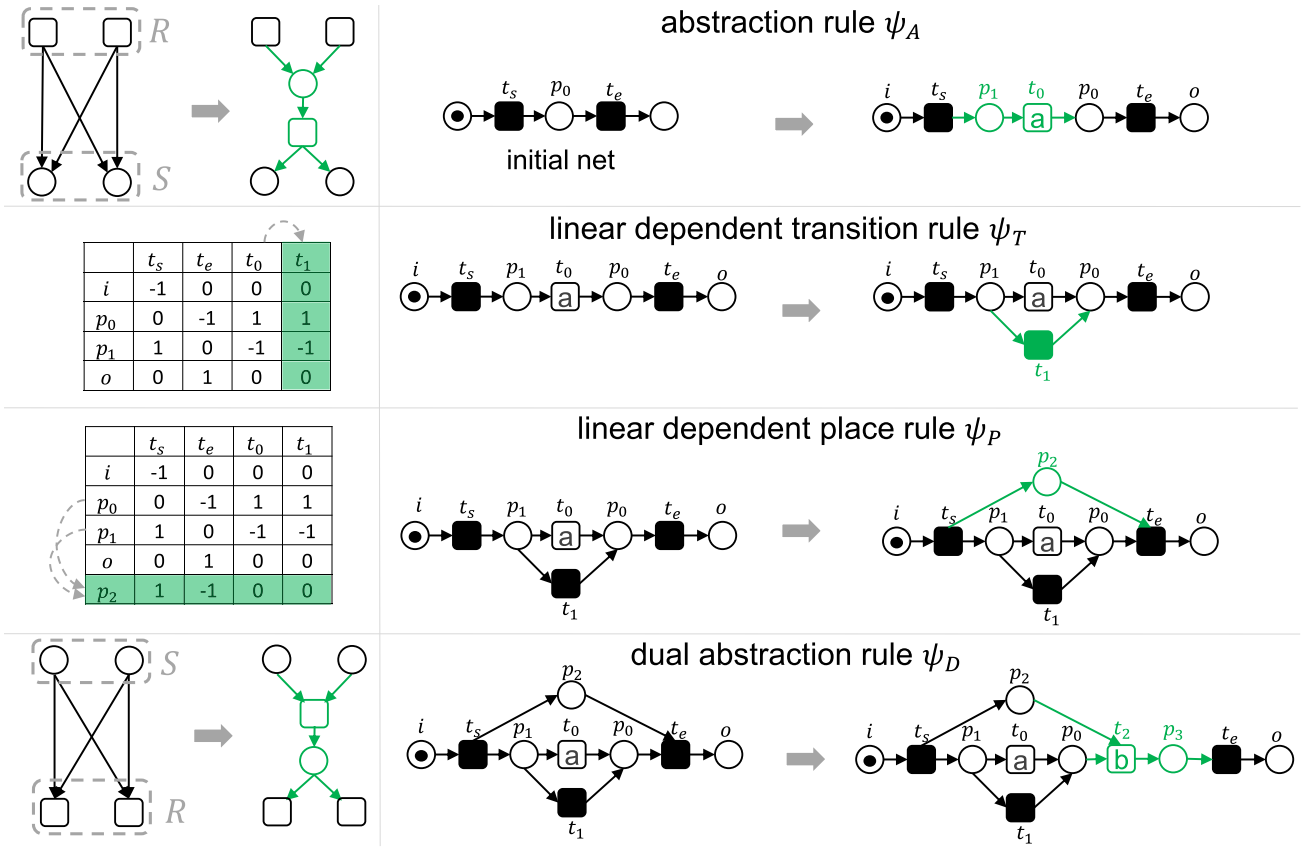


Fig. 2 Examples of the synthesis rules applications, where the highlighted parts (in green) indicate the newly added components

and $F' = F \cup \widetilde{F}$ where $\widetilde{F} \subseteq ((P \times \{t\}) \cup (\{t\} \times P))$, and $\forall_{t \in T \cap T'} l(t) = l'(t)$.

- Abstraction Rule ψ_A: $(W, W') \in \psi_A$ if (1) there exists a set of transitions $R \subseteq T$ and a set of places $S \subseteq P$ such that $(R \times S \subseteq F) \wedge (R \times S \neq \emptyset)$. (2) $SC(W')$ is constructed by adding an additional place $p \notin P$ and a transition $t \notin T$ such that $P' = P \cup \{p\}$, $T' = T \cup \{t\}$, $F' = (F \setminus (R \times S)) \cup ((R \times \{p\}) \cup (\{p\} \times \{t\}) \cup (\{t\} \times S))$, and $\forall_{t \in T \cap T'} l(t) = l'(t)$.
- Dual Abstraction Rule ψ_D: $(W, W') \in \psi_D$ if (1) there exists a set of places $S \subseteq P$ and a set of transitions $R \subseteq T$ such that $S \times R \subseteq F \wedge S \times R \neq \emptyset$. (2) W' is constructed by adding an additional transition $t \notin T$ and a place $p \notin P$ such that $P' = P \cup \{p\}$, $T' = T \cup \{t\}$, $F' = (F \setminus (S \times R)) \cup ((S \times \{t\}) \cup (\{t\} \times \{p\}) \cup (\{p\} \times R))$, and $\forall_{t \in T \cap T'} l(t) = l'(t)$.

The four rules preserve both the soundness and the free choice properties [2, 4]. Figure 2 (taken from [6]) provides examples of rule 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 Synthesis Miner

Having introduced the essential concepts, we are now ready to present the approach. Synthesis Miner employs an iterative procedure to modify an existing sound and free-choice WF-net. The algorithm operates in two modes:

1. Discovery: This mode directly discovers a WF-net from the corresponding event log [4], similar to standard process discovery algorithms. In this case, the existing model is an *initial net* [4] that initially contains only the artificial start and end transitions (as shown in Fig. 2). Subsequently, labeled transitions are added until every activity in the event log has a corresponding transition in the process model.
2. Adjustment: This mode adjusts an existing WF-net [7, 17]. Such a model can be discovered using other process discovery algorithms or provided by the stakeholders as a nominal model. In such a case, Synthesis Miner identifies and removes the problematic nodes from the existing WF-net before relocating them to a more suitable position.

Generally, during each iteration, both modes share three main steps in common: (1) prune the search space (by iden-

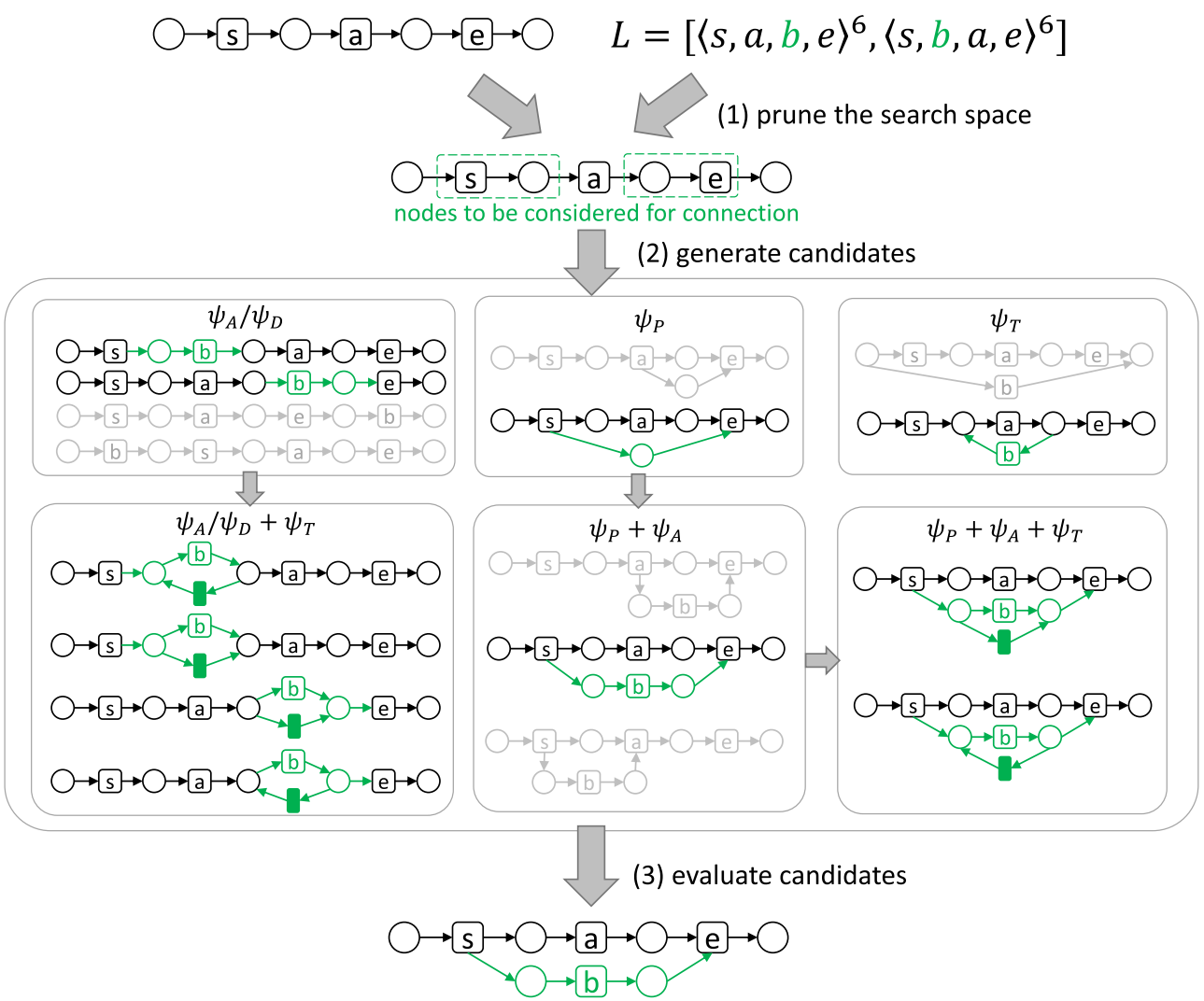


Fig. 3 An example of the generation and evaluation steps in Synthesis Miner. Initially, candidates are generated following predefined patterns [4] based on synthesis rules, as indicated by the symbols above each block. Next, each candidate undergoes evaluation using alignment-based conformance checking to calculate the F1-score. It should be noted that the figure does not display the full set of generated candidates

tifying the nodes to be modified) (2) generate candidates, and (3) evaluate candidates. Before diving into the details, we first describe these steps in general using the example illustrated in Fig. 3. In this figure, the addition of a transition labeled b to the existing net is depicted. First, we use log heuristics to prune the search space by locating the most probable existing nodes for connection. In the case shown, the log suggests that activity b follows activity s and precedes activity e, prompting Synthesis Miner to generate candidates that place the new node between the nodes labeled with s and e. Then, to generate candidates, Synthesis Miner tries to add a transition labeled b to the existing process model. Candidates with unlikely connections are ignored, as grayed-out examples are shown in Fig. 3 for illustration. For example, since the transition labeled a is concurrent to b, Synthesis Miner would not consider nets with connections to it. Using log heuristics helps to narrow down the search space of the best candidate.

Once the candidates are generated, they are evaluated based on their F1-score,[2] and the best-fit candidate is selected for the next iteration. In this example, the transition labeled b (with no option to skip or repeat) is inserted between the transitions s and e, resulting in a net that closely matches the

[2] Adopting the benchmark process discovery research [8], we consider the F1-Score as the target metric, given its definition as the harmonic mean of precision and recall. However, the target metric for selecting the candidate is a parameter that can be customized according to the user's preference.

log data (high fitness) and excludes any unseen behaviors (high precision). In the following, we detail each step of the Synthesis Miner.

4.1 Pruning the search space

There are many possible ways to add a new node to an existing WF-net using the synthesis rules (Definition 9). However, not all modifications are of interest, and evaluating all possibilities is not feasible. The main issue is that the search space (i.e., the number of possible candidate connections) increases exponentially as more nodes are considered for connection to the newly added node. In previous work [4, 5], log-based heuristics are employed to reduce the search space. Specifically, the process involves defining certain log properties that help to determine which activities precede and follow a given activity.

Definition 10 (Log Properties [4]) Let $L \in \mathcal{B}(\mathcal{A}^*)$ and $a, b \in \mathcal{A}$.

- $\#(a, L) = \Sigma_{\sigma \in L} |\{1 \leq i \leq |\sigma| \mid \sigma(i) = a\}|$ is the times a occurred in L.
- $\#(a, b, L) = \Sigma_{\sigma \in L} |\{1 \leq i < |\sigma| \mid \sigma(i) = a \wedge \sigma(i+1) = b\}|$ 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).
- $A^p_\theta(a, L) = \{a_p \in \mathcal{A} \mid caus(a_p, a, L) \geq \theta\}$ is the set of a's preceding activities, determined by threshold θ.
- $A^f_\theta(a, L) = \{a_f \in \mathcal{A} \mid caus(a, a_f, L) \geq \theta\}$ is the set of a's following activities, determined by threshold θ.

We use the example log L_s to help the readers understand the concepts defined in Definition 10.

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

In L_s, activity e is immediately followed by f 30 times, i.e., $\#(e, f, L_s) = 30$. On the other hand, $\#(f, e, L_s) = 0$ since activity f was never followed by e in log L_s. With these two values, one can calculate the causality $caus(e, f, L_s) = \frac{30-0}{30+0+1} = 0.968$. Using a default threshold value[3] of $\theta = 0.9$ [4], we conclude that activity f is in the set of e's following activities, i.e., $f \in A^f_{0.9}(e, L_s)$.

After identifying these sets of preceding and following activities as per Definition 10, their respective transitions can also be mapped.[4] A straightforward solution [4] is to assume that all nodes on the path from the set of preceding transitions to the set of following transitions can connect to the new node.

For example, in Fig. 4, a transition labeled e needs to be added to the net. Since activity e is preceded by a and followed by f in L_s (Definition 10), 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 need to check the linear dependency of 3^3=27 vectors because there are three possible connections $(-1, 1, 0)$ for each place in p_2, p_3, p_9. Additionally, 3^2=9 vectors must be checked when using the linear-dependent place rule (ψ_P) to add a place. These rules are used to build predefined patterns. This example shows that the number of candidates grows exponentially as more nodes are considered for connection.

Heuristic I

To further reduce the search space [7], we propose to only consider the sets of preceding/following transitions (Definition 10) and their post/pre-sets (Definition 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 $T_p \cup T_f \cup T_p \bullet \cup \bullet T_f$. In the following, we denote the set of nodes pending connection as $V_s = T_p \cup T_f \cup T_p \bullet \cup \bullet T_f$. The assumption is that the other nodes are 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.

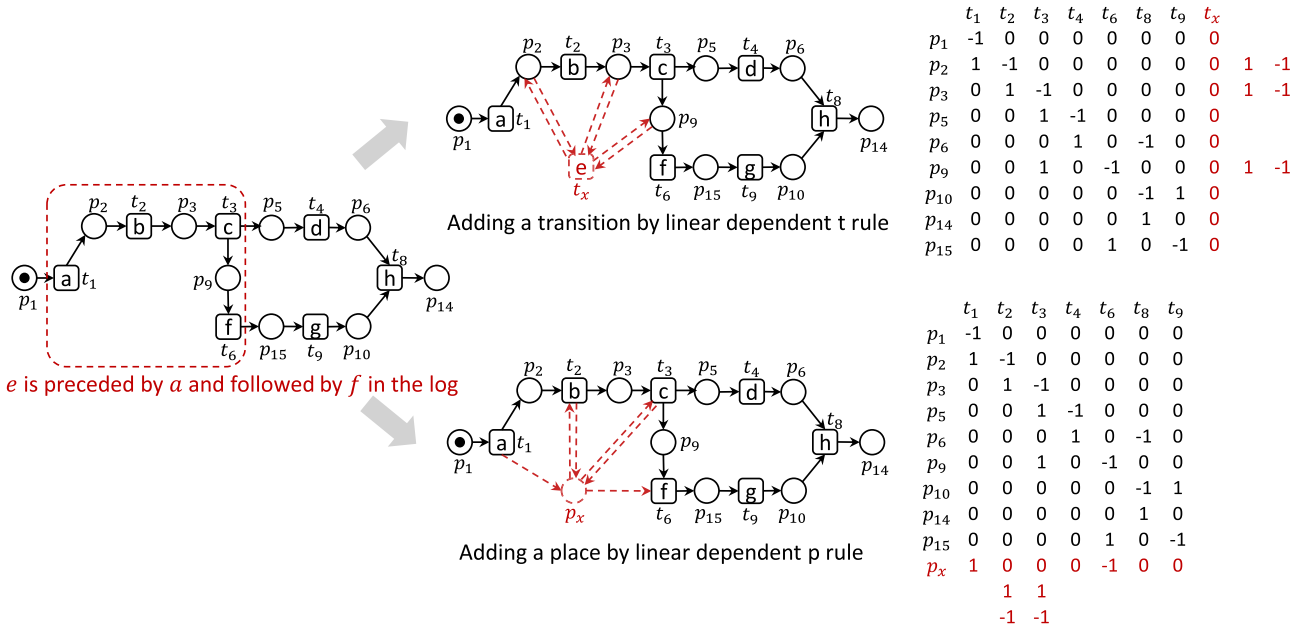
Using the example in Fig. 5 to illustrate the idea, we know that activity e is preceded/followed by activity a/f respectively in log L_s according to Definition 10. The set of nodes considered for connection is then $\{t_1, p_2, p_9, t_6\}$ as opposed to $\{t_1, p_2, t_2, p_3, t_3, p_9, t_6\}$ from the original strategy shown in Fig. 4. As shown in Fig. 5, the number of vectors pending for the 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.

4.2 Generating candidates

Once the nodes (set V_s) considered for connection are identified, the next step is to add a transition labeled with the

[3] In subsequent sections, a default threshold value of $\theta = 0.9$ [4] is used to define the preceding and following activities.

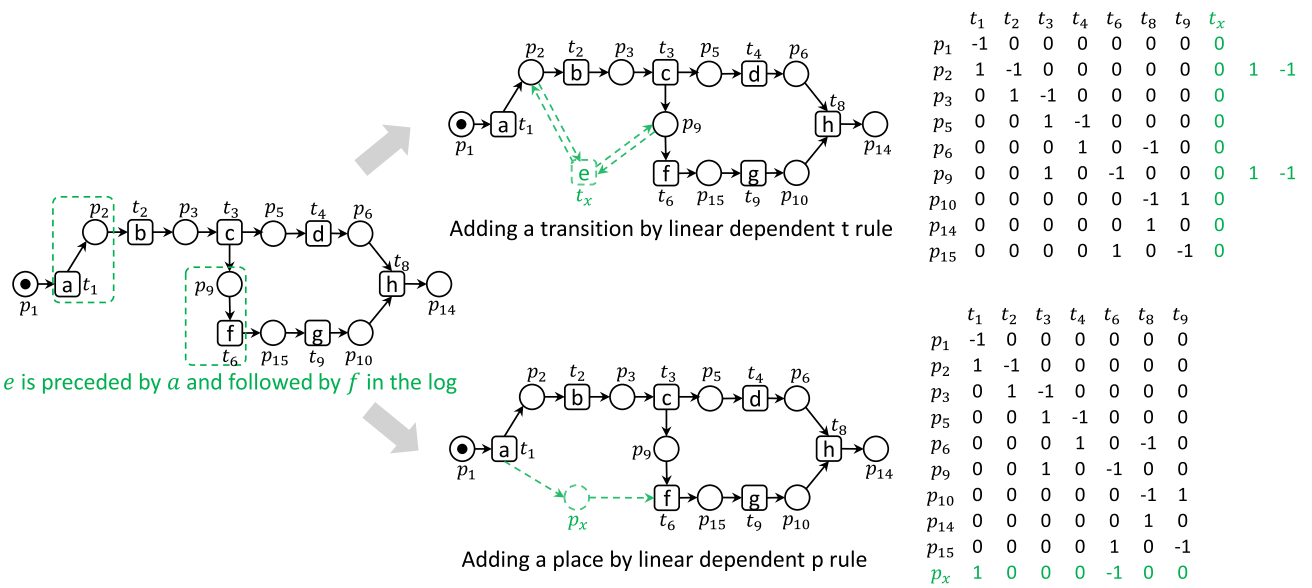
[4] For simplicity, we refer to these transitions as preceding and following transitions corresponding to the activity labels.



	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. 4 Adding a transition or a place using the linear dependent rules. The rows and columns beyond the incidence matrix represent the three potential connections between the newly added and existing nodes. For example, when examining the set of nodes t_1, p_2, t_2, p_3, t_3, p_9, t_6 along the path from activity a to activity f, linear dependency must be verified across $3^3+3^2=36$ vectors. Note that only the fundamental linear dependency rules are depicted in this figure. In practice, the optimal candidate (not shown here) combines the linear dependent place rule ψ_P and the abstraction rule ψ_A



	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. 5 Considering only the sets of preceding/following transitions 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 $S = \{p_x\}$

corresponding activity with various connections. In this section, we introduce the patterns used to add transitions for newly added activities. First, we discuss the application of a single synthesis rule. Three out of the four synthesis rules add a transition by definition: the linear dependent transition rule (ψ_T), the abstraction rule (ψ_A), and the dual abstraction rule (ψ_D). The only exception is the linear dependent place rule (ψ_P), which adds only a place to the existing model. Hence, in our approach, every application of ψ_P is immediately followed by an application of ψ_A to incorporate a transition. For example, the rule ψ_A is applied between the newly added place p' and its preset $\bullet p'$. This is valid because, by definition, every transition in $\bullet p'$ is connected to the place p', thereby satisfying the precondition of ψ_A. To be more precise, we define the rule combination, ψ_{PA}, that describes the pattern.

Definition 11 (Combined Rule ψ_{PA}) Let $W{=}(P, T, F, l)$ and $W''{=}(P'', T'', F'', l'')$ be free-choice workflow nets.

$(W, W'')\in\psi_{PA}$ if (1) $\exists_{W'=(P',T',F',l')}(W, W')\in\psi_P \wedge (W', W'')\in\psi_A$ and (2) $\exists!_{p^*\in P''}(\{p^*\}{=}P'\backslash P)\wedge(((T''\backslash T')\times\{p^*\})\subset F'')\wedge((T'\times\{p^*\})\not\subset F'')$.

To get the set of initial candidate nets, Synthesis Miner applies rules ψ_T, ψ_A, ψ_D, and ψ_{PA} to the potential nodes for connections V_s, derived from subsection 4.1. A concrete example can be seen in Fig. 3 in the corresponding annotated blocks. The next step is to add more patterns based on the log heuristics. In total, we identify four basic and local patterns for a transition, determined by whether it is repeatable and skippable. We start by introducing various pattern-building functions.

Definition 12 (Pattern-Building Functions [4]) Let $W = (P, T, F, l)$ and $W' = (P', T', F', l')$ be two free-choice workflow nets. Let $a \in \mathcal{A}$ be an activity label and $t_a \in T : l(t_a) = a$ be the corresponding transition in W. We define the three pattern-building functions[5] as

- $skip(W, a) = W'$ such that
  - $(W, W') \in \psi_T$
  - $F' = F\cup(\{t'\}\times t_a\bullet)\cup(\bullet t_a\times\{t'\})$ (where $t'\in T'\backslash T$)
  - $l' = l$ (t' is a silent transition)
- $loop_s(W, a)$ is defined by two cases:
  1. if $\nexists_{t^*\in((t_a\bullet)\bullet)}(|\bullet t^*| > 1)\wedge(\bullet t^*\backslash t_a\bullet \neq \emptyset)$, then $loop_s(W, a) = W'$ such that
     - $(W, W') \in \psi_T$
     - $F' = F \cup (t_a\bullet \times \{t'\}) \cup (\{t'\} \times \bullet t_a)$ (where $t' \in T'\backslash T$)
     - $l' = l$ (t' is a silent transition)
  2. otherwise, return $loop_s(W', a)$ such that
     - $(W, W') \in \psi_A$
     - $((\{t_a\} \times (P'\backslash P)) \in F') \wedge ((\{t_a\} \times P) \notin F')$
     - $l' = l$
- $loop_\tau(W, a)$ is defined by two cases:
  1. if $\nexists_{t^*\in((t_a\bullet)\bullet)}(|\bullet t^*| > 1)\wedge(\bullet t^*\backslash t_a\bullet \neq \emptyset)$, then $loop_\tau(W, a){=}W'$ such that
     - $(W, W') \in \psi_T$
     - $F' = F \cup (t_a\bullet \times \{t'\}) \cup (\{t'\} \times \bullet t_a)$ (where $t' \in T'\backslash T$)
     - $l' = (l\backslash\{(t_a, a)\})\cup\{(t', a)\}$ (the labels of t_a and t' are swapped)
  2. otherwise, return $loop_\tau(W', a)$ such that
     - $(W, W') \in \psi_A$
     - $((\{t_a\} \times (P'\backslash P)) \in F') \wedge ((\{t_a\} \times P) \notin F')$
     - $l' = l$

To be precise, Definition 12 may appear complicated, but in practice, it simply extends the candidate pool for the newly added transition. Given a workflow net W and an activity label a, the function *skip* adds a silent transition that shares the same connections as the corresponding transition t_a (associated with the input activity label). This modification follows the linearly dependent transition rule ψ_T, as the added silent transition is merely a duplication (and thus linearly dependent) of t_a. The function $loop_s$ behaves similarly, but with reversed connections relative to t_a, creating a loop. Lastly, $loop_\tau$ "swaps" the labels of t_a and a new transition t' to place activity a within an optional loop. The second case in the loop functions, namely $loop_\tau$ and $loop_s$, is necessary to preserve the free-choice property in certain edge cases. These extranets enable a transition to become skippable or repeatable—either as a strict loop or a non-strict one.

Using the running example shown in Fig. 6, for the transition (labeled e) added to the existing model, the synthesis miner adds three additional candidate nets to the pool. However, this straightforward (naive) method increases the computation needed for both the generation and evaluation steps. Additionally, we can often determine from the log whether an activity should be skippable or repeatable. To do this, we define two indicators with the corresponding thresholds.

Definition 13 (Pattern variables) Let $L \in \mathcal{B}(\mathcal{A}^*)$ and $a \in \mathcal{A}$ be an activity. Let $\theta_{skip}, \theta_{loop} \in [0, 1]$.

- $\#_1(a, L) = \frac{|\{\sigma\in L|a\in\sigma\}|}{|L|}$ is the Presence Indicator that represents the proportion of traces in the log L that contain activity a.

[5] In Definition 12, the input/output notations ($\bullet$) refer to the input net W. The superscript is omitted for readability.

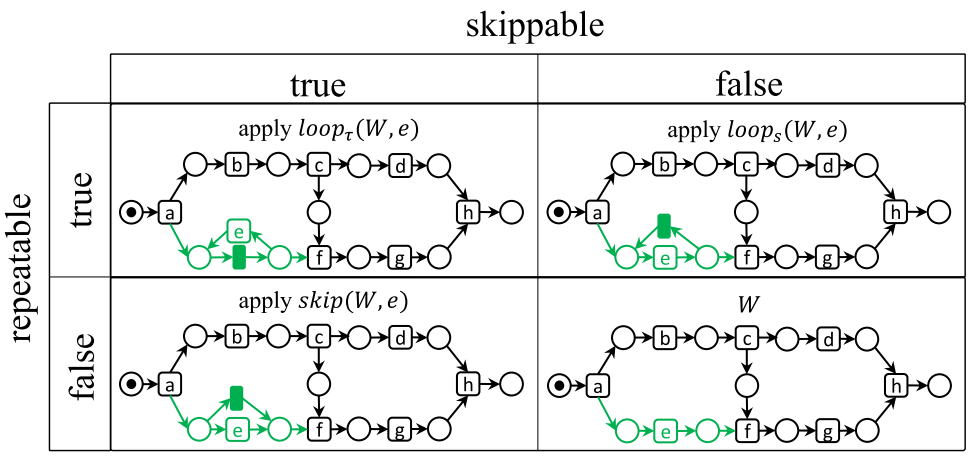


Fig. 6 An example showing how the pattern building functions are applied to create the different patterns for a labeled transition added to the existing net

– $\#_2(a, L) = \frac{|\{\sigma \in L \mid |\{1 \leq i \leq |\sigma| \mid \sigma(i) = a\}| \geq 2\}|}{|L|}$ is the Repetition Indicator that represents the proportion of traces in L in which a appears at least twice.

If $\#_1(a, L) > \theta_{skip}$, the Synthesis Miner treats activity a as non-skippable. Similarly, if $\#_2(a, L) > \theta_{loop}$, loop patterns are applied. In the running example log L_s, we observe that $\#_1(e, L) = 1$ and $\#_2(e, L) = 0$. With a threshold of 0.9, activity e is identified as neither skippable nor repeatable. As a result, only one pattern is generated for this candidate net. This targeted approach reduces the number of patterns that need to be generated by up to 75%, improving the runtime of both the generation and evaluation phases.

4.3 Evaluating candidates

The evaluation of the candidates is relatively straightforward. To evaluate a candidate net, we can use the corresponding projected log to assess the quality of the model, which is measured based on the widely adopted F1-score in process discovery research [8]. However, we observe that the modification often only affects a sub-part of the model in previous work [4–7].

Heuristic II

Motivated by such an 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 Sect. 4.1. To be precise, we first define the concept of a subnet in the context of this paper.

Definition 14 (Subnet) Let $N = (P, T, F)$ be a WF-net that is sound and free-choice. Let i be the source place, i.e., $i \in P : {\bullet i} = \emptyset$ and o be the sink place, i.e., $o \in P : o{\bullet} = \emptyset$. $N_s = (P_s, T_s, F_s)$ 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 $P_{in} \subseteq P \setminus P_s, P_{out} \subseteq P \setminus P_s, T_{start} \subseteq T_s, T_{end} \subseteq T_s$ 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}.
  - $\forall_{p \in P_s}\left({}^N{\bullet p} \cup p{\bullet}^N \subseteq T_s\right)$, all places in P_s only connects to transition in T_s.
  - $\forall_{t \in T_s \setminus T_{start}}\left({}^N{\bullet t} \subseteq P_s\right)$, except for T_{start}, the input places of every transition in T_s are in P_s.
  - $\forall_{t \in T_s \setminus T_{end}}\left(t{\bullet}^N \subseteq P_s\right)$, except for T_{end}, the output places of every transition in T_s are in P_s.
  - $\{i, o\} \cup P_s = \emptyset$, the source and sink places are not in the subnet (to exclude border cases).

The criteria of a subnet in Definition 15 are visualized in Fig. 7a, 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}. Figure 7b shows the transformed WF-net. Note that the WF-net transformed from the 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.

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.

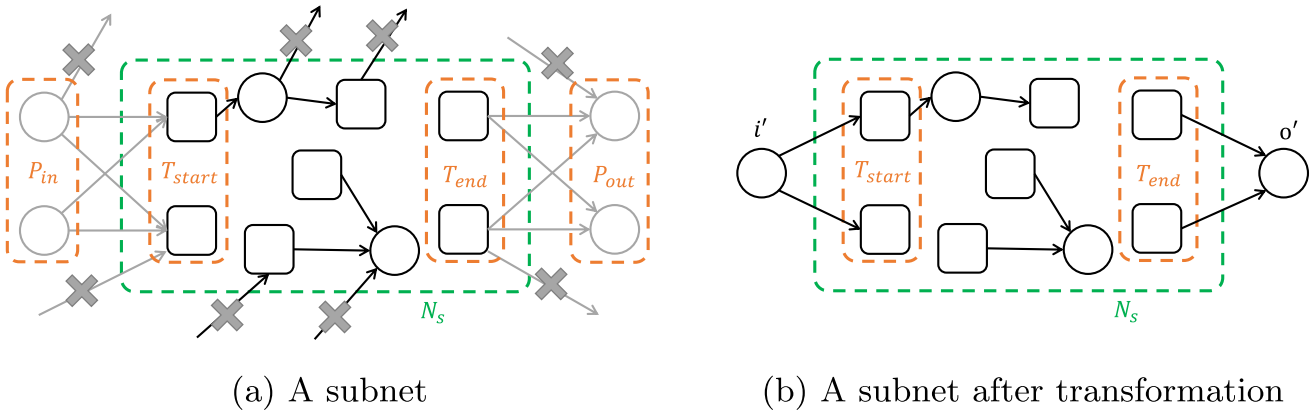


Fig. 7 An example showing the criteria of a subnet as defined in Definition 15 and the transformed WF-net

Definition 15 (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 and the patterns introduced in Sect. 4.2 can be used to generate candidates. The benefits of extracting minimal subnets are twofold.

1. Smaller incidence matrix: applying linear-dependent t/p rules (c.f. Definition 6) requires linear dependency checkup using Gaussian elimination,[6] 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 a conformance check on a smaller net and a smaller log also leads to a faster computation.

[6] As discussed in Sect. 4.2, the connection of a new node is randomly assigned to the other nodes. Thus, it might be possible that a generated candidate net is not sound anymore. To ensure soundness according to Definition 9, a linear-dependency check needs to be done.

To illustrate the idea, consider the running example as shown in Fig. 8, where the transition labeled 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 (Definition 10), 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. 8 (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. 8. Moreover, the conformance checking can also be decomposed. Specifically, all candidates derived from modifications on the minimal subnet are evaluated using the projected log $L_s{\restriction}_{\{d,f,g,h\}} = [\langle d, f, g, h\rangle^{40}, \langle f, g, d, h\rangle^{30}, \langle f, d, g, h\rangle^{30}]$. Then, the best candidate is selected and connected back to the original net. As shown in Fig. 8, 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.

Lastly, the modified net becomes the input for the next iteration and undergoes the three steps introduced above. In the *discovery* mode, this process continues until every activity in the log has a corresponding transition in the process model. In *adjustment* mode, the iteration stops when every labeled transition has been relocated. Additionally, every intermediate model is preserved, as they are all variations of the complete model (containing transitions for all activities). Once the iteration is complete, the model with the best F1-score is chosen.

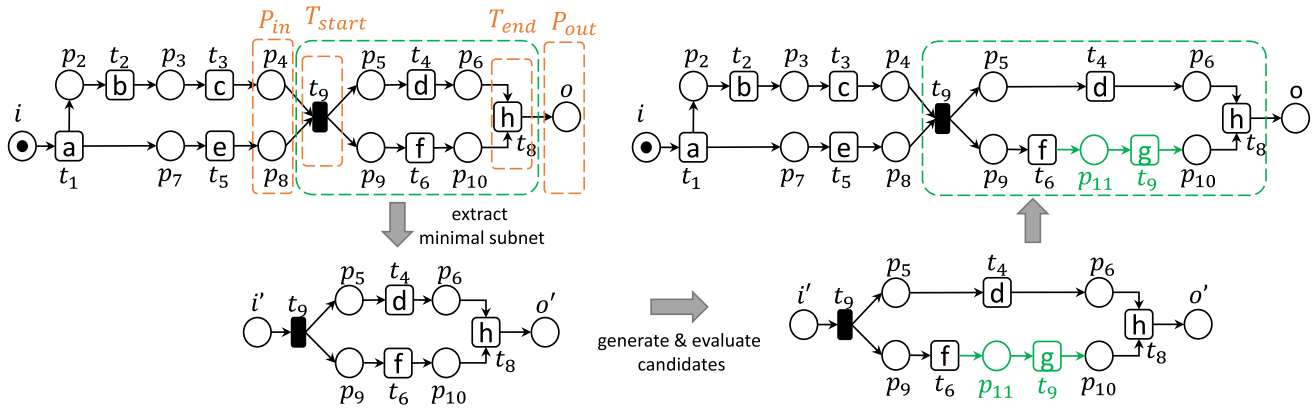


Fig. 8 As activity g is preceded by activity f and followed by activity h in log L_s, the minimal subnet can be extracted. The green dotted line highlights the extracted subnet, where $P_{in} = \{p_4, p_8\}$, $P_{out} = \{o\}$, $T_{start} = \{t_9\}$, $T_{end} = \{t_8\}$

5 Experimental evaluation

In this section, we present the experiments[7] conducted to evaluate the effectiveness of Synthesis Miner. The first experiment aims to investigate the heuristics while the second experiment compares the quality of the discovered models with two prominent discovery algorithms, namely Inductive Miner [3] and Split Miner [11].

Dataset For the evaluation, we use a collection of publicly available event logs from the *4TU Centre for Research Data*.[8] The BPI Challenge (BPIC), the Road Traffic Fines Management Process (traffic), and the SEPSIS logs are considered. The event logs cover various domains such as finance, software development, health care, and publich sectors. For the BPIC logs, we exclude those without explicit business processes (BPIC2011 and BPIC2016), as indicated in [8].

Since the proposed approach is sensitive to excessive noise, we exclude logs that require extensive filtering to yield a meaningful process with sufficient quality as stated in [8], e.g., BPIC2014, BPIC2015, BPIC2019, etc. Including such logs would not only distort the discovered structures but also confound the evaluation by introducing artifacts unrelated to the discovery capability itself. Exceptions are BPIC2012 and BPIC2017, as the prefixes of the activities clearly indicate different subprocesses. Therefore, the BPIC2012 and BPIC2017 logs are further divided into sublogs–BPIC2012A, BPIC2012O, BPIC2012W, BPIC2017A, BPIC2017O, and BPIC2017W–based on event prefixes, to capture distinct process variations. The statistics of the event logs are summarized in Table 1.

Table 1 Descriptive statistics of logs

Log	Events	Cases	Activities	Variants
BPIC2012A	73,936	13,087	11	17
BPIC2012O	36,259	5015	8	168
BPIC2012W	91,729	9658	8	2263
BPIC2017A	271,104	31,509	11	102
BPIC2017O	236,844	42,995	9	16
BPIC2017W	211,948	31,509	8	475
Helpdesk	27,292	4217	10	27
SEPSIS	15,214	1050	16	846
Traffic	561,470	150,370	11	231

For each event log in Table 1, we used the Inductive Miner—Infrequent (IMf) [3] and Split Miner (SM) [11] to create the initial models.

To enrich the input data, we use two different filter values (0.2 and 0.4) for IMf while default parameters for SM. The resulting model-log pairs for our experiments[9] are shown in Table 2. Each model-log pair is identified by the format [log name]_[IMf filter/SM].

5.1 Experiment I

The objective of the heuristics is to improve the time performance of the process discovery approach based on the synthesis rules introduced in [4, 5], while maintaining similar model quality. Therefore, the first experiment compares computation time and model quality with and without the proposed heuristics, with particular emphasis on the generation and evaluation steps.

[7] The complete setup and code for the experiment, including heuristics, are available at https://git.rwth-aachen.de/tsunghao.huang/fast-and-sound-heuristic.

[8] https://data.4tu.nl/.

[9] Note that in Table 2, we exclude the cases where the discovery algorithm fails to discover a model. Additionally, the helpdesk_04 from IMf is the same as helpdesk_02 so we only keep one.

Table 2 Descriptive statistics of models

Model	Fitness	Precision	F1	Transitions	Places	Arcs
BPIC2012A_02	0.997	0.637	0.777	16	14	36
BPIC2012A_04	0.978	0.754	0.851	14	14	32
BPIC2012A_sm	0.946	1.000	0.972	13	13	28
BPIC2012O_02	0.949	0.557	0.702	15	13	34
BPIC2012O_04	0.898	0.565	0.694	12	12	28
BPIC2012O_sm	0.860	1.000	0.925	8	7	16
BPIC2012W_02	0.866	0.860	0.863	22	14	44
BPIC2012W_04	0.482	0.649	0.553	8	6	16
BPIC2012W_sm	0.901	0.919	0.910	24	16	48
BPIC2017A_02	0.999	0.936	0.967	17	12	34
BPIC2017A_04	0.948	0.999	0.973	11	9	22
BPIC2017O_02	0.997	0.907	0.950	10	7	20
BPIC2017O_04	0.957	1.000	0.978	9	7	18
BPIC2017W_02	0.911	0.867	0.889	17	12	34
BPIC2017W_04	0.864	0.822	0.842	10	8	20
helpdesk_02	0.972	0.953	0.962	14	12	30
SEPSIS_02	0.903	0.581	0.707	24	23	62
SEPSIS_04	0.421	0.852	0.563	22	22	52
SEPSIS_sm	0.790	0.980	0.875	33	19	66
traffic_02	0.981	0.546	0.702	22	18	50
traffic_04	0.862	0.752	0.803	20	20	48

Instead of running through all iterations, we compare one iteration at a time by relocating every labeled transition from the existing net in Table 2. For each relocation, each heuristic is individually enabled or disabled to create four experimental configurations.

- $original$: The original Sythesis Miner as in [4, 5]
- $h1$: Only Heuristic 1 (Sect. 4.1) is enabled.
- $h2$: Only Heuristic 2 (Sect. 4.3) is enabled.
- $h1 + h2$: both heuristics are enabled.

In other words, each iteration is computed four times using the configurations. Since the heuristics are designed to speed up the generation and evaluation processes, we measured the time taken for these two steps. Specifically, for each process model, we removed each labeled transition and then added it back, recording the time required to generate and evaluate candidate nets in each case. Finally, we assessed and documented the quality of the resulting models.

5.1.1 Time reduction

Table 3 presents the computation times (in seconds) for generating and evaluating candidate nets. These times represent the average needed to add each labeled transition to the respective process model. As mentioned, there are four different configurations depending on the application of heuristics. The configurations are as follows: *h1* (only heuristic 1 is applied), *h2* (only heuristic 2 is applied), *h1+h2* (both heuristics are applied), and *original* (no heuristics, following the approach outlined in [5]).

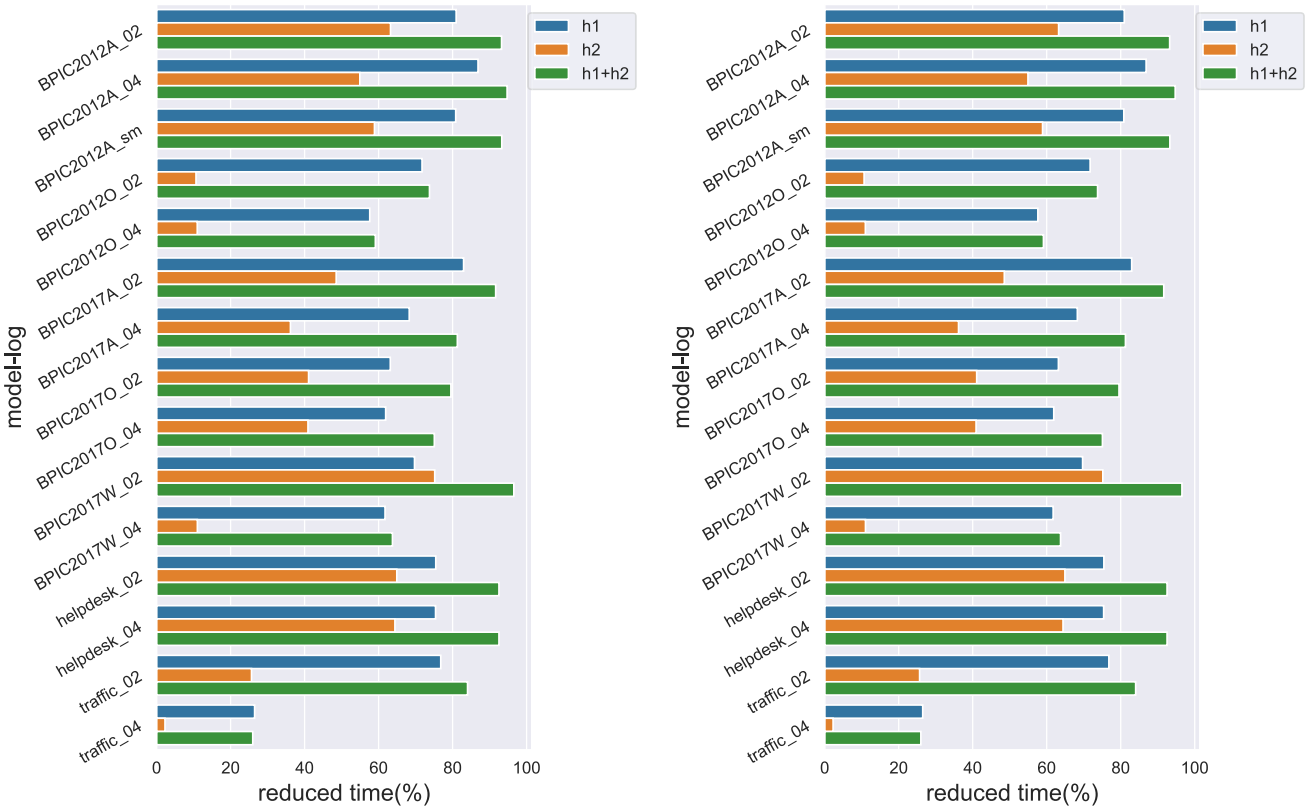
Table 3 shows a clear reduction in computation time for generating and evaluating candidate nets when heuristics are used.[10] In most cases, applying either heuristic *h1*, *h2*, or both together (*h1+h2*) leads to much faster results than the original approach from [5]. Some models benefit from larger reductions than others. For example, BPIC2012A_04 shows an extreme reduction in generation time from 165.989 s (original) down to 0.031 s (*h1+h2*) while traffic_04 has more moderate improvement reductions, with a generation time that goes from 6.678 s (original) to 0.453 s (*h1+h2*).

Figure 9 shows a clearer indication of how much time (%) is reduced compared to the original approach [5]. As shown in Fig. 9a, Heuristic 1 (*h1*) generally performs better than Heuristic 2 (*h2*) alone for generation times in most models. As indicated by the green bars in Fig. 9, the combination *h1+h2* generally achieves the lowest computation times,

[10] To have a meaningful comparison, we only include the model-log pairs for which the Synthesis Miner was able to produce results for all four configurations on time (timeout = 1800 s). Therefore, all the iterations for BPIC2012W and SEPSIS logs are not included in the resulting table due to the high variants in the log.

Table 3 Computation time (seconds) for generation and evaluation of candidate nets

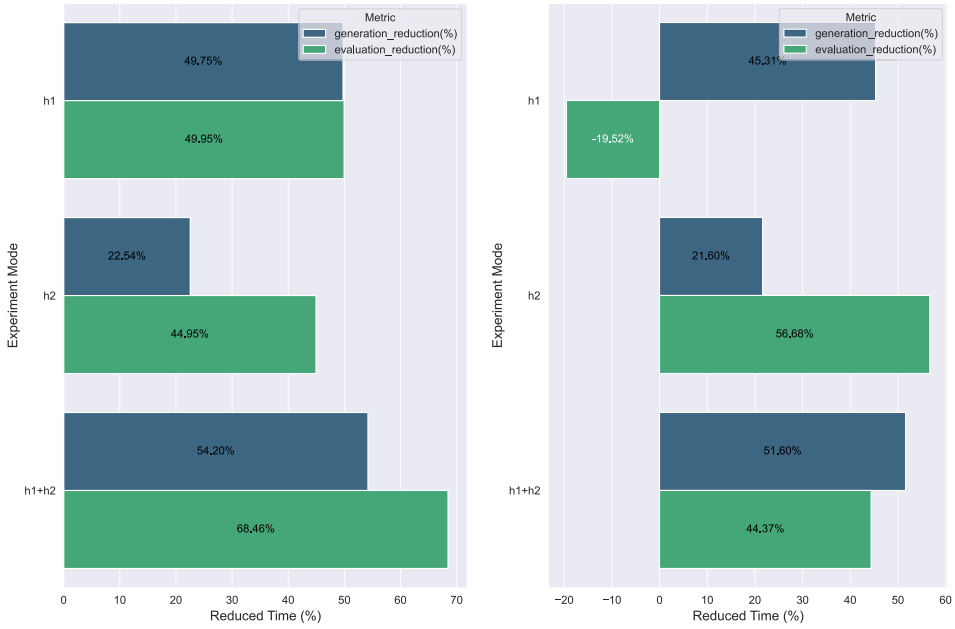
Model-Log	Generation				Evaluation			
	h1	*h2*	*h1+h2*	*Original*	*h1*	*h2*	*h1+h2*	*Original*
BPIC2012A_02	0.050	42.212	0.040	50.935	1.758	3.396	0.625	9.233
BPIC2012A_04	0.036	155.688	0.031	165.989	1.403	4.817	0.563	10.687
BPIC2012A_sm	0.049	46.099	0.039	53.962	1.483	3.183	0.520	7.740
BPIC2012O_02	0.022	1.057	0.022	1.186	7.537	23.827	7.001	26.667
BPIC2012O_04	0.016	0.356	0.016	0.365	7.596	15.948	7.325	17.917
BPIC2017A_02	0.036	16.343	0.023	18.247	6.374	19.256	3.129	37.418
BPIC2017A_04	0.019	0.776	0.012	0.667	7.689	15.473	4.538	24.241
BPIC2017O_02	0.023	0.145	0.016	0.174	4.727	7.559	2.625	12.829
BPIC2017O_04	0.021	0.106	0.016	0.136	5.052	7.837	3.308	13.264
BPIC2017W_02	0.012	0.070	0.008	0.207	23.527	19.280	2.675	77.657
BPIC2017W_04	0.009	0.149	0.009	0.140	16.050	37.328	15.214	41.953
helpdesk_02	0.030	2.876	0.022	3.744	0.918	1.312	0.279	3.740
traffic_02	0.353	51.896	0.305	59.913	24.951	79.961	17.163	107.551
traffic_04	0.417	7.435	0.453	6.678	44.905	59.703	45.215	61.091



(a) time reduced in generation

(b) time reduced in evaluation

Fig. 9 Bar charts showing the time reduced (%) compared to the original approach [5]



(a) Time reduced (%) (b) Normalized time reduced (%)

Fig. 10 Bar charts showing the time reduced (%) compared to the original approach [5] aggregated by experiment mode (*h1, h2, h1+h2*). (**b**) is generated by dividing the generation/evaluation time by the number of candidate nets

demonstrating that the combined heuristics can reduce processing times even further than each applied individually.

One can notice that Heuristic 2 (*h2*) alone does not necessarily have a lower generation time. For example, in both BPIC2017W_04 and traffic_04, the original approach has a lower generation time than Heuristic 2. This result can be explained by the computation time of extracting a minimal subnet and projecting the sublog, which are also included in the generation time calculation. However, the benefit of doing so proves to be beneficial in the evaluation stage, as both logs show a lower evaluation time for Heuristic 2 than the original setting. The comparison of overall reduced time (compared to the original) across all model-log pairs also confirms such benefits, as visualized in Fig. 10a.

The results and discussion so far suggest that Heuristic 1 ($h1$) has a stronger effect on reducing computation than Heuristic 2 ($h2$). However, it is important to note that the computation time heavily depends on the size of the candidate pool. The main effect of $h1$ is to reduce the number of candidates. In contrast, $h2$ aims to reduce the workload of generating and evaluating individual candidates. Additionally, upon closer examination of Heuristic 2 (*h2*), it becomes clear that the minimal subnet extracted is not always smaller than the original WF-net. This could be due to the structure of the model or the specific nodes considered, which may require expansion across the entire WF-net. For example, in Fig. 5 the nodes considered for connection are limited to t_1, p_2, p_9, t_6. However, by Definition 15, it is impossible in this case to identify a smaller subnet distinct from the entire WF-net.

To provide a fairer comparison and investigate the impact of Heuristic 2 (*h2*) more precisely, we focus on scenarios where *h2* can yield a smaller subnet and compare these with the same model-log pairs where *h2* remains inactive. This leaves us with 76% of the data. Furthermore, the generation and evaluation time is normalized by dividing the number of candidates generated before calculating the reduced time (%). The result of the analysis is visualized in Fig. 10b. As expected, Fig. 10b shows that for a single candidate, $h2$ is more efficient at reducing computation time for both generation and evaluation. Additionally, $h1$ is expected to reduce generation time since computation grows exponentially with the number of candidates.

To our surprise, $h1$ appears to have a negative impact on the evaluation of an individual candidate. A closer look at the corresponding cases suggests that this is due to the two-stage evaluation used by the Synthesis Miner. The first stage calculates the fitness score of each candidate in the pool, and only those with a fitness score above the threshold survive to the next stage for precision evaluation. In this setting, most of the candidates generated by the original approach would be ruled out at the first stage, while the candidates generated by $h1$ are more likely to move to the next stage. In such cases, more conformance checks have to be performed for more candidates in the second stage. In simple terms, the quality of the candidates generated by $h1$ is generally better, which means that the evaluation takes longer per candidate.

In summary, the experiments show that both heuristics improve time performance, with Heuristic 1 ($h1$) generally leading to more significant reductions in computation time

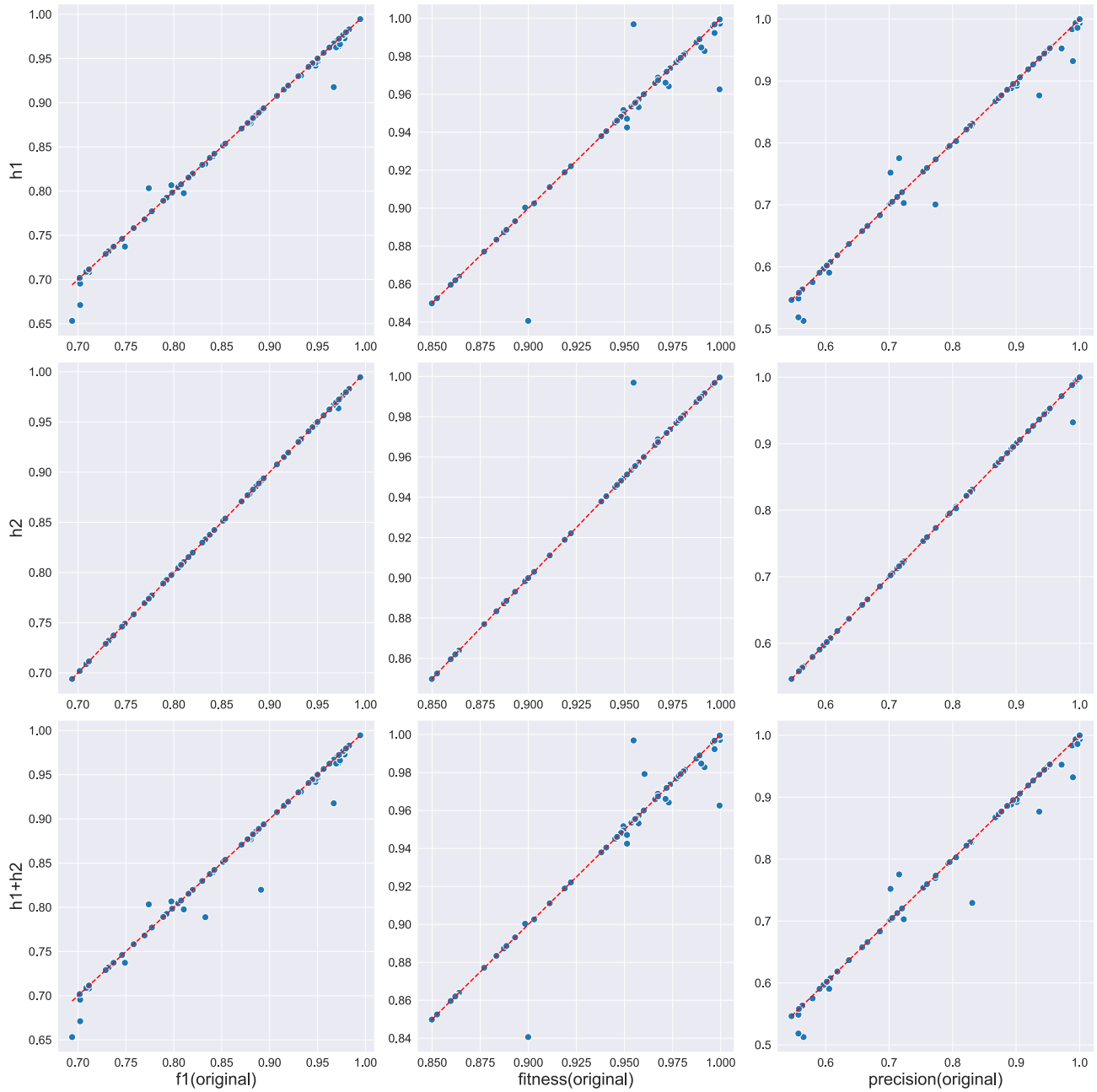


Fig. 11 Scatter plots comparing the quality (F1) of the resulting models using different settings (*h1*, *h2*, *h1+h2*). The red dashed lines indicate the ideal situation where the quality of the resulting models with/without applying heuristics is the same

compared to Heuristic 2 ($h2$). While $h1$ reduces the number of candidates, $h2$ optimizes the generation and evaluation of individual candidates. The combination of both heuristics ($h1+h2$) provides the best results in terms of time reduction. However, $h2$ alone is more effective in the evaluation time, despite not always reducing the generation time.

5.1.2 Model quality trade-off

To compare the quality of the process models with and without applying heuristics, we analyzed F1-score, fitness, and precision across different configurations. Figure 11 presents scatter plots for each metric. For each point, the x-axis shows the metric of the original approach and the y-axis shows the same metric when using one of the heuristics (*h1*, *h2*, or

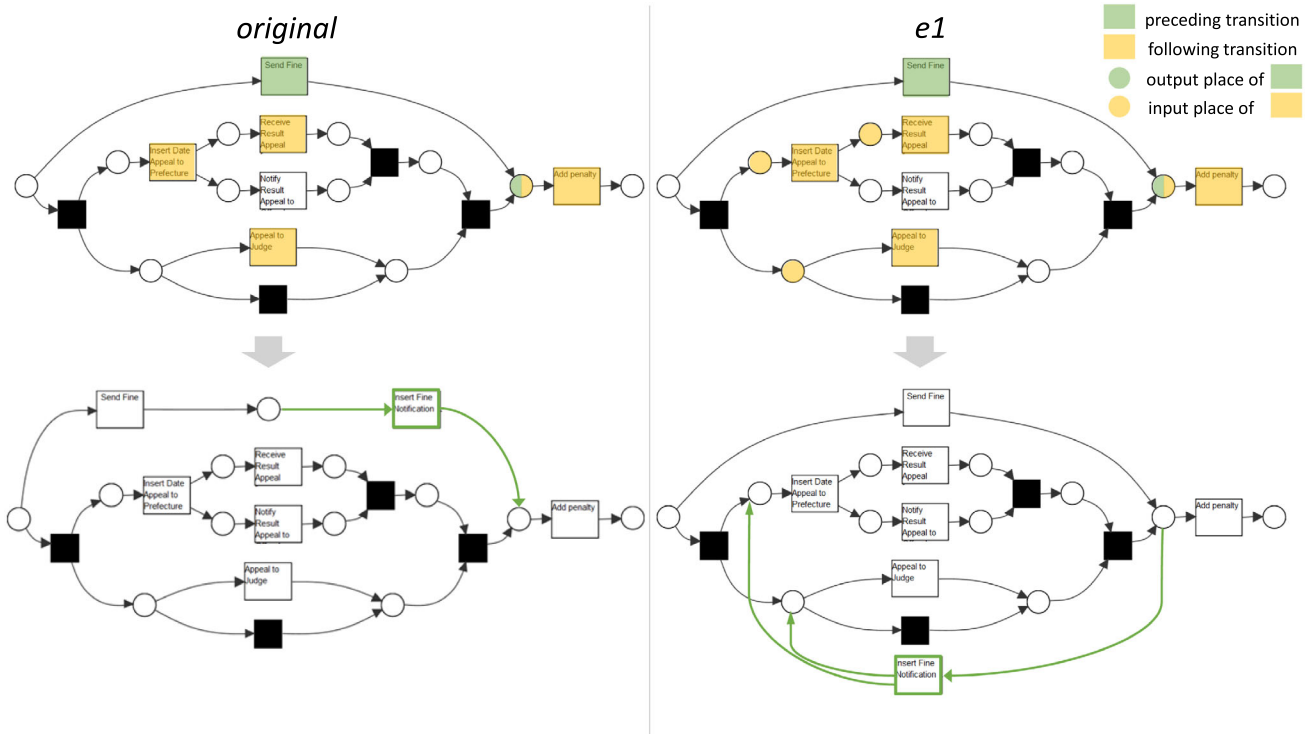


Fig. 12 Examples showing parts of the process model for the model-log pair *traffic_04*, where transition labeled *Insert Fine Noftification* should be relocated

h1+h2). The red dashed line represents an ideal outcome, where the quality is identical with and without the heuristic.

We discuss the quality aspect along the three quality metrics.

- *F1-score* Among the heuristics, *h2* maintains F1-scores closest to the original, with data points aligning almost exactly along the ideal line. This suggests that *h2* preserves the quality of the model better than other heuristics, ensuring minimal change from the original. *h1* and *h1+h2* also perform reasonably well for F1, but show slightly more variation around the ideal line, which means that there is a small risk of quality fluctuations with these settings. This might indicate that *h2* provides a more stable F1 performance across different models.
- *Fitness* The fitness results similarly show that *h2* closely followed the original scores, with almost all points staying close to the ideal line. This indicates that *h2* maintains the fitness of the model. Both *h1* and *h1+h2* show a bit more spread around the line with points sometimes below the line, indicating some variation in fitness.
- *Precision* The precision plots show similar patterns, which also highlight *h2* as the best choice, with points consistently close to the ideal line.

We now take a closer look at the quality differences of each configuration by discussing a few cases where the metrics significantly deviated from those of the original.

Heuristic 1 (*h1*) usually reduces the number of nodes considered for connections, so it offers a trade-off between processing time and model quality. Interestingly, there are cases where *h1* actually improves model quality compared to the original approach. Figure 12 illustrates one such case. The original method aims to find nodes along the path between the preceding and following transitions. However, in the model shown in Fig. 12, there is rarely a direct path between them because the preceding transition (labeled *Send Fine*) is independent (part of a choice structure) from most of the following transitions. The only extra node is the place between the transitions labeled *Send Fine* and *Add penalty*. In contrast, *h1* includes not only the sets of preceding and following transitions but also their post- and pre-sets (places), as shown on the right in Fig. 12. Heuristic 1 (*h1*) works well in such a case when the model behavior deviates significantly from the corresponding log.

While *h2* generally keeps F1-scores close to the original, some points still show noticeable deviations. A deeper look reveals that this difference often comes from changes in the ranking of candidate models, which vary depending on whether subnet extraction is used. Using Fig. 13 as an example, Synthesis Miner aims to relocate the transition labeled *O_Accepted*, highlighted in green. When using the original

configuration, the algorithm chooses the model in Fig. 13a because its overall F1-score is higher than that of the alternative in Fig. 13a. However, when *h2* is enabled, the model in Fig. 13b is preferred, as its extracted subnet has a higher F1 score than the subnet in Fig. 13a. This variation helps explain why F1-scores for *h2* sometimes differ from the original configuration. Lastly, the configuration *h1+h2* combines the effect of both heuristics and thus inherits the deviations in the quality metrics from both.

In summary, *h2* appears to be the most reliable heuristic for maintaining the model's original quality across all metrics (F1, fitness, and precision), showing minimal fluctuation from the original scores. While *h1* and *h1+h2* can also preserve quality reasonably well, they introduce more variability.

5.2 Experiment II

The second experiment compares the model quality from Synthesis Miner to the two other prominent discovery algorithms, namely Inductive Miner and Split Miner.

We adopt a setting similar to the first experiment to investigate if it can improve the quality of the existing models. Specifically, we apply Synthesis Miner to relocate problematic transitions for every model-log pair, aiming to improve the quality of the models.

The results in Table 4 suggest that applying Synthesis Miner can lead to modest yet meaningful improvements in model quality for several model-log pairs. In particular, we observe gains in precision, and consequently in F1-score, for several models originally discovered using both Inductive Miner and Split Miner. For instance, in the case of BPIC2012A_02, precision improved from 0.637 to 0.797, resulting in a notable increase in F1-score, despite a small drop in fitness. Indeed, Inductive Miner is known for producing process models with high fitness but low precision. However, this trade-off is not always present when using Synthesis Miner. In other examples—such as BPIC2012O_02 and BPIC2017O_02/04—both fitness and precision improved, leading to better overall model quality. Interestingly, BPIC2017O_02 and BPIC2017O_04 led to the same adjusted model, despite differences in their respective input models.

That said, there are also several cases (e.g., BPIC2012O_sm, BPIC2017A_04, BPIC2017W_02/04) where Synthesis Miner did not yield any improvement and simply returned the original model. Moreover, we excluded some results (BPIC2012W and SEPSIS) because Synthesis Miner failed to return a result within the timeout limit (1800 s), likely due to the high number of variants in the event logs. Another important limitation is the significantly high computation time required by Synthesis Miner. While other discovery methods typically complete within a few seconds, Synthesis Miner can take several minutes or more to produce a result.

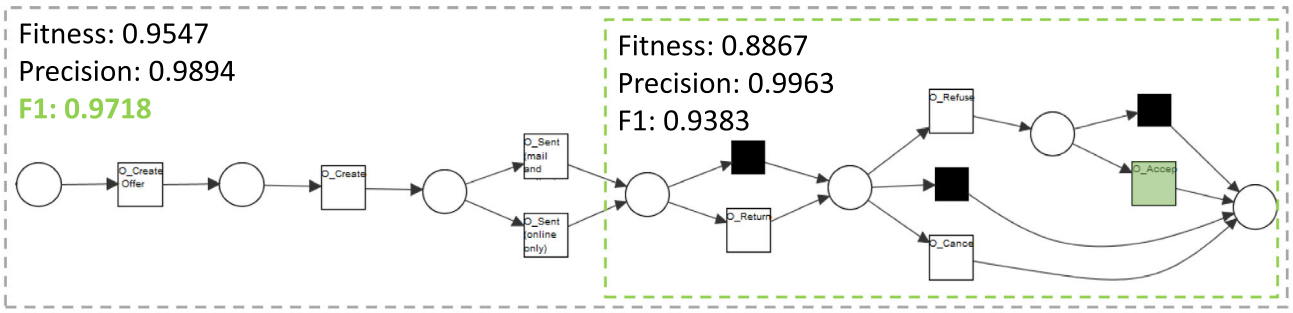
In summary, while Synthesis Miner demonstrates the potential to enhance process model quality in certain scenarios, its effectiveness is not universal, and its computational demands are non-trivial. These findings suggest that the proposed approach may complement existing discovery techniques, but it should be applied selectively, particularly when runtime is a concern. Further work is needed to improve scalability and to develop heuristics that can help determine when such adjustments are likely to be beneficial.

6 Threats to validity and limitations

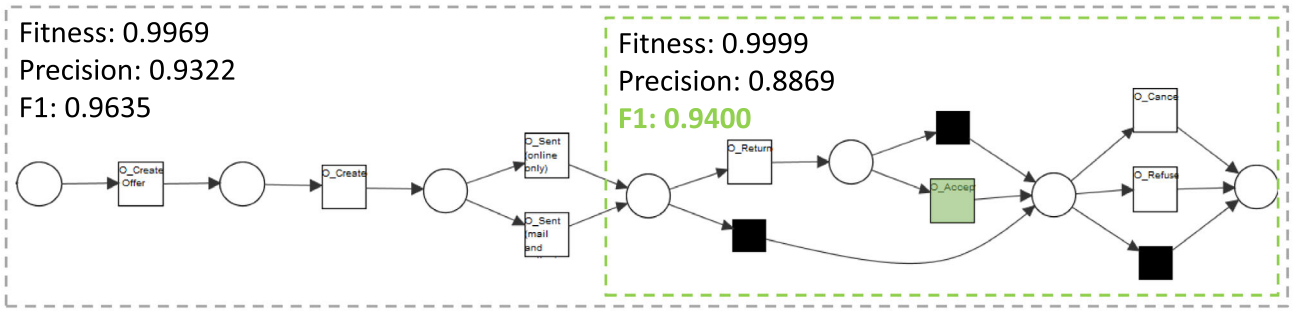
While the proposed extensions to the Synthesis Miner demonstrate significant improvements in scalability and efficiency, it is important to reflect on potential threats to validity and limitations to provide a critical yet constructive perspective.

6.1 Threats to validity

1. *Generalizability to diverse logs* The evaluation uses a set of publicly available event logs that represent a range of processes. While these logs are diverse, their ability to fully capture the variability of real-world scenarios remains an open question. Logs with highly complex behavior or uncommon patterns may expose additional challenges not observed in the current study.
2. *Dependency on heuristics* The proposed log-based pruning and minimal subnet extraction rely on heuristic thresholds (e.g., causality thresholds) to guide the operation. These thresholds perform well in our experiments, but slight adjustments may be needed to optimize performance for other datasets. This could introduce additional effort in tuning for specific scenarios.
3. *Evaluation metrics focus* Using F1-score as the primary evaluation metric allows for an effective balance between fitness and precision. However, it may not fully capture nuanced trade-offs or other quality aspects valued in specific contexts. Complementary metrics could provide further insights into the impact of the proposed extensions.
4. *Noise sensitivity* Another important threat to validity concerns the sensitivity of the proposed approach to noisy event data. Since Synthesis Miner relies on log statistics to guide the application of synthesis rules, excessive noise can distort causal relations and lead to misleading candidate structures. To mitigate this risk in our evaluation, we excluded logs that required extensive filtering to yield a meaningful process with sufficient quality. While this choice avoids artifacts introduced by heavy preprocessing, it may also bias the evaluation toward



(a) Resulting model when using the original approach (without *h1* and *h2*)



(b) Resulting model when using Heuristic 2 to extract the minimal subnet

Fig. 13 Comparison of model quality on the whole net and the extracted subnet. The model-log pair is BPIC2017O_02 and Synthesis Miner aims to relocate the transition labeled *O_ACCEPTED*, highlighted in green

Table 4 Model quality comparison

Model	Before Fitness	Precision	F1	Time	After Fitness	Precision	F1	Time
BPIC2012A_02	0.997	0.637	0.777	4	0.966	0.797	0.873	60
BPIC2012A_04	0.978	0.754	0.851	3	–	–	–	63
BPIC2012A_sm	0.946	1.000	0.972	5	0.961	0.997	0.979	535
BPIC2012O_02	0.949	0.557	0.702	5	0.962	0.914	0.937	358
BPIC2012O_04	0.898	0.565	0.694	5	0.816	0.844	0.829	247
BPIC2012O_sm	0.860	1.000	0.925	3	–	–	–	129
BPIC2017A_02	0.999	0.936	0.967	16	0.967	0.999	0.983	1467
BPIC2017A_04	0.948	0.999	0.973	15	–	–	–	244
BPIC2017O_02	0.997	0.907	0.950	12	0.984	0.994	0.989	438
BPIC2017O_04	0.957	1.000	0.978	12	0.984	0.994	0.989	433
BPIC2017W_02	0.911	0.867	0.889	32	–	–	–	733
BPIC2017W_04	0.864	0.822	0.842	46	–	–	–	1279
helpdesk_02	0.972	0.953	0.962	6	0.972	0.993	0.982	16
traffic_02	0.981	0.546	0.702	89	0.982	0.635	0.771	994
traffic_04	0.862	0.752	0.803	82	0.859	0.892	0.875	1184

cleaner datasets. As a result, the reported results primarily reflect the performance of the approach under conditions where noise is limited. A systematic investigation of noise robustness is left for future work.

6.2 Limitations

1. *Computational overhead in specific scenarios* Although the extensions significantly improve computational efficiency, their impact may be limited when the extracted subnet is large or closely resembles the original WF-net. In such cases, the overhead introduced by extraction and transformation may offset the intended performance gains, as observed in Sect. 5.
2. *Model structure constraints* The approach assumes that input models are sound and free-choice. While this aligns with the design goals of Synthesis Miner, it may restrict applicability in settings with unstructured or noisy event logs that do not conform to these assumptions.
3. *Impact on candidate pool quality* Extension 1 narrows the candidate pool using log heuristics, which may lead to the exclusion of potentially useful candidates. This introduces a trade-off between computational efficiency and the completeness of the search space, potentially affecting model optimality.
4. *Scalability* As shown in Experiment 2, Synthesis Miner can still exhibit long computation times, especially for complex logs with high variant diversity. This is expected as the proposed approach rely on conformance checking in the intermediate iterations to select the best model for each iteration. While it can produce competitive results, the high computational cost may limit its practical use in time-constrained or large-scale settings.

7 Conclusion

Synthesis Miner algorithm [4, 5] is developed to discover models featuring non-block structures while maintaining desirable theoretical guarantees (free-choiceness and soundness) using synthesis rules based on free-choice net theor [2]. Adopting an iterative setting, the approach aims to find the best modification (w.r.t. model quality) to an existing workflow net by generating and evaluating various candidate nets. However, the generation and evaluation steps, involving synthesis rules and alignment-based conformance checking, respectively, presented significant scalability challenges.

To address these challenges, this paper proposed heuristic extensions that leverage the insight that modifications often impact only a subpart of the model. By isolating these subparts, we introduced methods to accelerate the process. Specifically, log heuristics were applied to narrow the search space further, and the generation and evaluation steps were broken into smaller tasks by focusing on minimal subnets of the process model. This design allows the algorithm to retain its theoretical guarantees while improving its practical usability in larger and more complex discovery tasks.

The first experiment showed that these enhancements led to substantial reductions in computation time, particularly when both heuristics were used in combination. Model quality remained comparable to the original version, suggesting that the scalability improvements do not compromise quality too much. The results show the trade-off between efficiency and robustness, as the improvements mainly target runtime while preserving accuracy.

The second experiment assessed the model quality produced by Synthesis Miner against two widely used discovery algorithms: Inductive Miner [3] and Split Miner [11]. While Synthesis Miner did not improve quality in all cases, it generated models with competitive—and occasionally superior—precision and F1-scores. These results demonstrate the potential of the approach as a flexible alternative, especially when structural expressiveness and formal guarantees are important. Nonetheless, the high computation time in some scenarios indicates that further improvements in scalability are still necessary.

These contributions enhance the scalability of the Synthesis Miner and extend its applicability to larger and more complex process discovery tasks. By addressing computational bottlenecks, this work ensures that the algorithm remains both efficient and reliable for discovering non-block structured models. At the same time, the study highlights limitations that need to be acknowledged: scalability remains a concern for very large logs, and noise sensitivity constrains applicability in certain domains.

Future research could focus on several enhancements to the Synthesis Miner algorithm. Refining the narrowing of the candidate pool, such as incorporating place fitness evaluations, could enable earlier filtering of non-viable candidates, reducing computational overhead and streamlining generation. Expanding log heuristics to capture specific behavioral patterns like concurrency, loops, or rare events might further improve accuracy in pinpointing optimal transition positions, potentially eliminating the need for exhaustive evaluations. Additionally, testing the algorithm on models discovered by diverse techniques beyond Inductive Miner could reveal its adaptability to more complex, unstructured, or noisy logs, broadening its applicability in real-world scenarios. Finally, a promising avenue lies in integrating noise-handling strategies and investigating incremental or parallelized evaluation to further balance scalability, accuracy, and robustness.

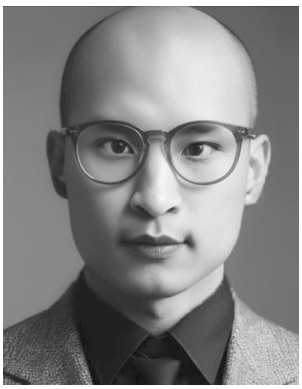
In summary, this work represents a step forward in improving the efficiency and usability of the Synthesis Miner algorithm while maintaining the formal guarantees of soundness and free-choice properties.

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

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Aalst, W.M.P., Carmona, J. (eds.): Process Mining Handbook. Lecture Notes in Business Information Processing, vol. 448. Springer, Cham, Switzerland (2022). https://doi.org/10.1007/978-3-031-08848-3
2. Desel, J., Esparza, J.: Free Choice Petri Nets, vol. 40. Cambridge University Press, Cambridge (1995)
3. Leemans, S.J.J.: Robust process mining with guarantees. PhD thesis, Technische Universiteit Eindhoven (2017)
4. Huang, T., Aalst, W.M.P.: Discovering sound free-choice workflow nets with non-block structures. In: Almeida, J.P.A., Karastoyanova, D., Guizzardi, G., Montali, M., Maggi, F.M., Fonseca, C.M. (eds.) Enterprise Design, Operations, and Computing - 26th International Conference, EDOC 2022, Bozen-Bolzano, Italy, October 3-7, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13585, pp. 200–216. Springer, Cham, Switzerland (2022). https://doi.org/10.1007/978-3-031-17604-3_12
5. Huang, T., Aalst, W.M.P.: Unblocking inductive miner—while preserving desirable properties. In: Aa, H., Bork, D., Proper, H.A., Schmidt, R. (eds.) Enterprise, Business-Process and Information Systems Modeling—24th International Conference, BPMDS 2023, and 28th International Conference, EMMSAD 2023, Zaragoza, Spain, June 12-13, 2023, Proceedings. Lecture Notes in Business Information Processing, vol. 479, pp. 327–342. Springer, Cham, Switzerland (2023). https://doi.org/10.1007/978-3-031-34241-7_23
6. Huang, T., Aalst, W.M.P.: Comparing ordering strategies for process discovery using synthesis rules. In: Troya, J., Mirandola, R., Navarro, E., Delgado, A., Segura, S., Ortiz, G., Pautasso, C., Zirpins, C., Fernández, P., Ruiz-Cortés, A. (eds.) Service-Oriented Computing - ICSOC 2022 Workshops - ASOCA, AI-PA, FMCIoT, WESOACS 2022, Sevilla, Spain, November 29 - December 2, 2022 Proceedings. Lecture Notes in Computer Science, vol. 13821, pp. 40–52. Springer, Cham, Switzerland (2022) https://doi.org/10.1007/978-3-031-26507-5_4
7. Huang, T., Schneider, E., Pegoraro, M., Aalst, W.M.P.: Fast & sound: Accelerating synthesis-rules-based process discovery. In: Aa, H., Bork, D., Schmidt, R., Sturm, A. (eds.) Enterprise, Business-Process and Information Systems Modeling - 25th International Conference, BPMDS 2024, and 29th International Conference, EMMSAD 2024, Limassol, Cyprus, June 3–4, 2024, Proceedings. Lecture Notes in Business Information Processing, vol. 511, pp. 259–274. Springer, Cham, Switzerland (2024). https://doi.org/10.1007/978-3-031-61007-3_20
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). https://doi.org/10.1109/TKDE.2018.2841877
9. Schuster, D., Zelst, S.J., Aalst, W.M.P.: Incremental discovery of hierarchical process models. In: Dalpiaz, F., Zdravkovic, J., Loucopoulos, P. (eds.) RCIS 2020. Lecture Notes in Business Information Processing, vol. 385, pp. 417–433. Springer, Cham, Switzerland (2020). https://doi.org/10.1007/978-3-030-50316-1_25
10. Buijs, J.C.A.M., Dongen, B.F., Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: CEC 2012, pp. 1–8. IEEE, Piscataway, NJ, USA (2012). https://doi.org/10.1109/CEC.2012.6256458
11. 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). https://doi.org/10.1007/s10115-018-1214-x
12. 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). https://doi.org/10.1016/J.DATAK.2018.04.007
13. Kourani, H., Zelst, S.J.: POWL: partially ordered workflow language. In: Francescomarino, C.D., Burattin, A., Janiesch, C., Sadiq, S. (eds.) Business Process Management - 21st International Conference, BPM 2023, Utrecht, The Netherlands, September 11-15, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14159, pp. 92–108. Springer, Cham, Switzerland (2023). https://doi.org/10.1007/978-3-031-41620-0_6
14. Dixit, P.M.: Interactive process mining. PhD thesis, Technische Universiteit Eindhoven (2019)
15. Dixit, P.M., Buijs, J.C.A.M., Aalst, W.M.P.: Prodigy: Human-in-the-loop process discovery. In: RCIS 2018, pp. 1–12. IEEE, Piscataway, NJ, USA (2018). https://doi.org/10.1109/RCIS.2018.8406657
16. Aalst, W.M.P.: The application of petri nets to workflow management. J. Circuits Syst. Comput. **8**(1), 21–66 (1998). https://doi.org/10.1142/S0218126698000043
17. Fahland, D., Aalst, W.M.P.: Model repair—aligning process models to reality. Inf. Syst. **47**, 220–243 (2015)



Tsung-Hao Huang is a research associate at the Chair of Process and Data Science, RWTH Aachen University. His research focuses on process discovery with formal guarantees and scalable conformance checking, with additional experience in process mining applications within logistics and supply chain management.

Enzo Schneider is a researcher and PhD candidate at the Institute for Highway Engineering (ISAC) at RWTH Aachen University, Germany. He received his Master's degree in Computer Science from RWTH Aachen, with a thesis in process mining. His current research focuses on urban climate prediction and advanced route planning to address challenges like urban heat island effect and improve urban sustainability. He also has prior industry experience as a data analyst.

Marco Pegoraro is a research assistant in the Chair of Process and Data Science at RWTH Aachen University. His main research activity pertains the extension of process mining techniques to uncertain, non-deterministic and probabilistic data. His other research interests are predictive process monitoring and the application of process mining to clinical and medical data, optimizing operations in healthcare settings.

Wil M. P. van der Aalst is an Alexander von Humboldt professor at RWTH Aachen University, leading the Process and Data Science (PADS) group. He is also the Chief Scientist at Celonis, part-time affiliated with the Fraunhofer FIT. His research interests include process mining, Petri nets, business process management, workflow automation, and data science. According to Research.com, he is the second-highest-ranked computer scientist in Germany and ranked 9th worldwide (2025 ranking). Van der Aalst is an IFIP Fellow, IEEE Fellow, and ACM Fellow, as well as an elected member of the Royal Netherlands Academy of Arts and Sciences, the Royal Holland Society of Sciences and Humanities, the Academy of Europe, and the German Academy of Science and Engineering.