Data & Knowledge Engineering 161 (2026) 102508

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Rule-guided process discovery☆

Ali Norouzifar [a,*], Marcus Dees [b], Wil van der Aalst [a]

[a] RWTH University, Aachen, Germany
[b] UWV Employee Insurance Agency, Amsterdam, Netherlands

ARTICLE INFO

Keywords:
Process mining
Process discovery
Domain knowledge

ABSTRACT

Event data extracted from information systems serves as the foundation for process mining, enabling the extraction of insights and identification of improvements. Process discovery focuses on deriving descriptive process models from event logs, which form the basis for conformance checking, performance analysis, and other applications. Traditional process discovery techniques predominantly rely on event logs, often overlooking supplementary information such as domain knowledge and process rules. These rules, which define relationships between activities, can be obtained through automated techniques like declarative process discovery or provided by domain experts based on process specifications. When used as an additional input alongside event logs, such rules have significant potential to guide process discovery. However, leveraging rules to discover high-quality imperative process models, such as BPMN models and Petri nets, remains an underexplored area in the literature. To address this gap, we propose an enhanced framework, IMr, which integrates discovered or user-defined rules into the process discovery workflow via a novel recursive approach. The IMr framework employs a divide-and-conquer strategy, using rules to guide the selection of process structures at each recursion step in combination with the input event log. We evaluate our approach on several real-world event logs and demonstrate that the discovered models better align with the provided rules without compromising their conformance to the event log. Additionally, we show that high-quality rules can improve model quality across well-known conformance metrics. This work highlights the importance of integrating domain knowledge into process discovery, enhancing the quality, interpretability, and applicability of the resulting process models.

1. Introduction

Information systems are widely implemented across organizations to support and manage their processes. These processes can range from order-to-cash workflows in supply chain management to billing in hospitals, university systems assisting students with course registration and examinations, or claim handling in insurance companies. Across these domains, actions performed by various resources are recorded in databases, generating event-based data. This data serves as the foundation for process mining, a field dedicated to discovering process models, conducting conformance checking, and optimizing processes. By leveraging event logs, process mining provides organizations with valuable insights to analyze and improve their operations effectively.

☆ This research was supported by the research training group "Dataninja" (Trustworthy AI for Seamless Problem Solving: Next Generation Intelligence Joins Robust Data Analysis) funded by the German federal state of North Rhine-Westphalia, Germany.

* Corresponding author.
E-mail addresses: ali.norouzifar@pads.rwth-aachen.de (A. Norouzifar), Marcus.Dees@uwv.nl (M. Dees), wvdaalst@pads.rwth-aachen.de (W.M.P. van der Aalst).

https://doi.org/10.1016/j.datak.2025.102508
Received 23 December 2024; Received in revised form 30 June 2025; Accepted 28 August 2025
Available online 2 September 2025

One of the core analyses in process mining is process discovery, which focuses on generating descriptive process models from event logs. These models enable organizations to understand how their processes operate, identify deviations from expected behaviors, and uncover inefficiencies such as bottlenecks. However, relying solely on event logs often results in models that lack alignment with domain knowledge or specific operational rules, limiting their practical utility and accuracy.

Incorporating supplementary information, such as process rules or domain knowledge, can significantly enhance the quality of discovered models. Declarative constraints, such as those expressed in the *Declare* language, are commonly used to capture process rules for modeling and conformance checking. Despite their widespread use in related tasks, leveraging such rules for discovering imperative models, like BPMN or Petri nets, remains under-explored. Recent efforts, such as [1], have demonstrated that integrating user-defined or discovered rules into process discovery workflows can improve model quality, but several challenges remain.

In this work, a *rule* refers to any specification that deterministically evaluates whether an observed process behavior is allowed or disallowed. Rules can be derived using various methods, including:

- **Automated Discovery:** Extracting rules from event logs using tools like Minerful or DeclareMiner to mine declarative constraints.
- **Normative Guidelines and Documentation:** Deriving rules from formal documentation, such as study plans specifying course sequences and prerequisites.
- **Natural Language Processing (NLP):** Extracting rules from process documentation or expert conversations using NLP techniques.
- **Expert Input:** Capturing rules directly from domain experts based on their knowledge and understanding of the process.

The IMr framework proposed in [1] integrates such rules into process discovery, starting with an event log and a set of user-defined or discovered rules. However, integrating rules into an inductive mining-based process discovery introduces some challenges. The representational bias of this algorithm favors the discovery of block-structured models, which, although easy to interpret and effective in many scenarios, are less flexible than declarative rules in capturing complex activity relations such as long-term dependencies. Additionally, process discovery algorithms must balance adherence to rules with generating interpretable and generalizable models.

In this paper, we extend the IMr framework introduced in [1] to address these challenges and enhance its theoretical, methodological, and practical contributions. Specifically, our work introduces the following innovations:

- A general mathematical foundation for integrating rules into process discovery, enabling support for a broader range of rule specification languages.
- Improvements to rule-handling heuristics, allowing the algorithm to identify promising cuts even when no rule-compliant options are available, rather than defaulting to rule-agnostic discovery.
- A new evaluation metric to measure rule deviations and their significance, offering a more comprehensive assessment of the alignment between discovered models and input constraints.
- An extended evaluation using various rule discovery configurations, comparing rule-based discovery with baseline methods (e.g., rule-agnostic approaches) and state-of-the-art techniques such as Split Miner.

The remainder of this paper is organized as follows. In Section 2, we review the existing literature relevant to our method. Section 3 introduces the notations and concepts foundational to our approach. In Section 4, we present the components of the IMr framework and describe how they interact to discover process models. Section 5 outlines the general foundation of our framework, highlighting its ability to incorporate diverse rule specifications. In Section 6, we focus on declarative constraints, proposing a solution for handling rule violations and selecting promising process structures. Section 7 details the event logs used in our experiments and presents real-world evaluations to demonstrate the benefits of our method compared to alternative approaches. Finally, in Section 8, we summarize the main contributions and conclusions of the paper.

2. Related work

Automatic process discovery has been extensively studied, yet fundamental challenges persist despite the multitude of proposed algorithms [2]. A key difficulty lies in achieving a balance among fitness, precision, simplicity, and generalization when modeling these processes [3]. Real-life processes are inherently complex, making the discovery of high-quality process models a challenging task.

To address this complexity, active trace clustering techniques [4] have been introduced, which divide event logs into more homogeneous subsets, enabling the discovery of specific models tailored to each group. Another approach involves abstracting infrequent behaviors from event logs to simplify the resulting process models [5]. Preprocessing event logs to remove noisy and infrequent behaviors is a widely adopted strategy to reduce unnecessary complexity and improve the quality of discovered models [6].

Existing process discovery frameworks predominantly focus on a single event log as input, often neglecting supplementary information sources that could enhance the discovery process. The incorporation of external information into process discovery has been explored in various forms [7]. Such additional information may include knowledge about desirable and undesirable aspects captured in separate event logs. Different research methodologies have been proposed to incorporate this information into the discovery of imperative process models [8,9] and declarative process models [10,11].

Process rules can either be discovered automatically, such as declarative rules, or provided by domain experts as additional input, offering valuable guidance on how process models should be structured. The approach proposed in [1] utilizes such rules alongside event logs to discover process models that better align with the specified rule constraints. While conformance metrics are effective for evaluating process models against a single event log, they fail to capture alignment with external information sources, such as domain knowledge or predefined rules. This limitation underscores the need for discovery and conformance checking approaches that integrate diverse input types, enhancing both the quality and relevance of the discovered models.

Another valuable source of information lies in the textual or natural language resources available within organizations. For instance, domain experts often possess an intuitive understanding of expected process behavior, which can be complemented by natural language documentation detailing process workflows. In [12], a large language model-based approach is introduced to encode natural language information and conversations into formalized rules. These rules can then guide process discovery, as demonstrated in [1], enhancing the alignment of discovered models with organizational knowledge.

The integration of supplementary information can also occur post-discovery to refine process models through repair [13], interactive enhancement [14], or performance optimization of the underlying processes [15]. Interactive process discovery provides an iterative mechanism to improve models with external guidance. For instance, the approach in [14] begins with an initial process model and utilizes user-selected trace variants to incrementally update and refine the model. While this method underscores the value of incorporating additional data, it is heavily dependent on the availability and quality of the initial model.

Even when additional information resources are not directly available, certain methods have been developed to enrich the inputs for process discovery. One such method involves the artificial generation of negative events to guide the discovery process, as demonstrated in [9]. These negative events represent prohibited behaviors inferred from observed data and help steer the discovery algorithm towards more accurate and meaningful models.

Declarative languages, such as Declare [16,17] and compliance rule graphs [18], provide interpretable frameworks for modeling process constraints. These languages are capable of representing complex relationships between activities, making them well-suited for capturing nuanced process behavior. Declarative process discovery and rule-based techniques leverage the expressiveness of these frameworks to gain deeper insights into process dynamics. Moreover, extensions to these approaches enable the incorporation of labeled traces, facilitating the discovery of declarative models that can effectively distinguish between different process variants [10,11].

In [19], declarative rules are applied post-discovery to refine process models. The method involves modifying the initially discovered model and selecting the best version that conforms to the predefined rules. In contrast, our approach integrates these rules directly into the discovery process, ensuring alignment with the rules from the very beginning.

Recursive discovery methods, such as the inductive miner, have been extensively extended to tackle various challenges in process discovery. For instance, the extension proposed in [20] incorporates filtering of infrequent behavior to generate more representative process models. Approaches like those in [8,21], and [22] enhance the discovery process by evaluating a large set of process structures in each recursion and selecting the most suitable one based on optimization objectives. Another known process discovery technique is the Split Miner, which serves as a state-of-the-art alternative for discovering high-quality process models [23]. In this work, we compare the models discovered by our proposed method with those generated by several discovery techniques from the literature.

The integration of additional information sources and rule-based guidance offers a promising avenue for advancing process discovery techniques. However, achieving a balance between external constraints and traditional conformance metrics remains a significant challenge, underscoring the need for innovative methodologies to align process models with diverse inputs. In this paper, we build upon the concept introduced in [1], extending the integration of a set of rules alongside an event log as inputs to guide process discovery and ensure alignment with predefined constraints.

3. Preliminaries

This section provides the formal definitions and mathematical notations used throughout the paper. These definitions form the foundation for understanding the proposed framework. A multiset extends the concept of a set by allowing multiple occurrences of the same element. $\mathcal{B}(A)$ represents the set of all multisets over a base set A. For example, $B_1 = [x^2, y]$ and $B_2 = [x, y, z^{10}]$ are two multisets over $A = \{x, y, z\}$. For $B \in \mathcal{B}(A)$, the frequency of an element $a \in A$ in B is denoted by $B(a)$. We write $b \in B$ if $B(b) > 0$. $|B| = \sum_{b \in B} B(b)$ is the number of elements in this multiset. Considering B_1 and B_2, we can write, $B_1(x) = 2$, $B_2(z) = 10$, $z \notin B_1$, and $|B_2| = 12$. The power set of a set A, denoted $\mathcal{P}(A)$, is the set of all subsets of A, including the empty set and A itself. A^* is the set of all finite sequences over A. We introduce an event log formally as a multiset of traces, i.e., a sequence of activities.

Definition 1 (*Event Log*)**.** Let $\mathcal{A}$ be the universe of activities. A trace $\sigma = \langle a_1, a_2, \dots, a_n \rangle \in \mathcal{A}^*$ is a finite sequence of activities. Each occurrence of an activity in a trace is an event. An event log $L \in \mathcal{B}(\mathcal{A}^*)$ is a multiset of traces. $\mathcal{L}$ is the universe of event logs.

For example, $\langle a, b, c \rangle$ represents a trace consisting of a, followed by b, and then c. Traces can be combined through concatenation; for instance, concatenating $\langle a, b \rangle$ with $\langle c, d \rangle$, denoted as $\langle a, b \rangle . \langle c, d \rangle$ results in $\langle a, b, c, d \rangle$. The length of a trace σ is indicated by $|\sigma|$, representing the number of elements in the trace. $\sigma(i)$ refers to the ith element of the sequence, where $1 \leq i \leq |\sigma|$.

Since we are building our framework based on the inductive mining technique, process tree notation is used as the representation. Process trees can be converted to Petri nets or BPMN models as other well-known process notations.

Definition 2 (*Process Tree Syntax*). Let $\bigoplus = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ be the set of process tree operators and let $\tau \notin \mathcal{A}$ be the so-called silent transition, then

- activity $a \in \mathcal{A}$ is a process tree,
- the silent activity τ is a process tree,
- let $M_1, \dots, M_n$ with $n > 0$ be process trees and let $\oplus \in \bigoplus$ be a process tree operator, then $\oplus(M_1, \dots, M_n)$ is a process tree.

$\mathcal{M}$ is the universe of all process trees we can define over the universe of activities $\mathcal{A}$.

Considering $M \in \mathcal{M}$, $act(M)$ extracts the set of activities used in the process tree such that $act(a) = a$, $act(\tau) = \emptyset$, and $act(\oplus(M_1, \dots, M_n)) = act(M_1) \cup \dots \cup act(M_n)$. $\mathcal{M}_\Sigma = \{M \in \mathcal{M} | act(M) = \Sigma\}$ is the set of all possible process trees generated over a set of activities $\Sigma \subseteq \mathcal{A}$.

In this paper, we use a slightly adapted version of regular expression to formalize the language of the models and perform mathematical operations. The symbol $\boldsymbol{a}$ represents a set $\{\langle a \rangle\}$ including a trace consisting of $a \in \mathcal{A}$. Given two expressions A and B, the operations are defined as follows: $A|B$ represents the union $A \cup B$, $A.B$ corresponds to the set of concatenated traces $\{a.b \mid a \in A \text{ and } b \in B\}$, and A^* denotes the set of traces formed by zero or more repetitions of traces in A. For instance,

- $\boldsymbol{a}.\boldsymbol{b} = \{\langle a, b \rangle\}$
- Considering $A = \{\langle a, b \rangle\}$ and $B = \{\langle c, d \rangle\}$, $A|B = \{\langle a, b \rangle, \langle c, d \rangle\}$
- $\boldsymbol{a}^* = \{\langle\rangle, \boldsymbol{a}, \boldsymbol{a}.\boldsymbol{a}, \boldsymbol{a}.\boldsymbol{a}.\boldsymbol{a}, \dots\}$

Each process tree operator $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$ has a semantic which generates a special type of behavior.

Definition 3 (*Semantics of Process Trees [24]*). Let $\Sigma \subseteq \mathcal{A}$ denote a subset of activities. The function $\phi : \mathcal{M}_\Sigma \rightarrow \mathcal{P}(\Sigma^*)$ extracts the set of traces allowed by a process tree which we refer to as the language of this process tree. The semantics of process trees can be defined as follows:

- For the silent activity ($\tau \notin \mathcal{A}$): $\phi(\tau) = \{\langle\rangle\}$
- For an atomic activity ($a \in \Sigma$): $\phi(a) = \{\langle a \rangle\}$
- For a composite operator $\oplus(M_1, \dots, M_n)$:

$$\phi(\oplus(M_1, \dots, M_n)) = \oplus_\phi(\phi(M_1), \dots, \phi(M_n))$$

Here, $\oplus_\phi$ refers to a *language-join function*, which is specific to each process tree operator. Let $T_1, \dots, T_n$ be sets of traces, with $n \geq 1$. The language-join functions for standard operators are defined as:

- **Sequence**: $\rightarrow_\phi (T_1, \dots, T_n) = T_1.T_2 \dots T_n$
- **Exclusive choice**: $\times_\phi(T_1, \dots, T_n) = T_1 \cup T_2 \cup \dots \cup T_n$
- **Concurrency**: $\wedge_\phi(T_1, \dots, T_n) = T_1 \circledast T_2 \circledast \dots \circledast T_n$
- **Loop**: $\circlearrowleft_\phi (T_1, \dots, T_n) = T_1.(\times_\phi(T_2, \dots, T_n).T_1)^*$

The shuffle product $T_1 \circledast \dots \circledast T_n$ takes sets of traces from $T_1, \dots, T_n$ and interleaves their traces $\forall_{1 \leq i \leq n} \sigma_i \in T_i$ while maintaining the order within each subtrace σ_i.

If a process tree consists of an operator as the root node and single activities as children, the language of it is as follows:

- $\rightarrow$ denotes the sequential composition of children, e.g., $\phi(\rightarrow (a, b)) = \{\langle a, b \rangle\}$.
- $\times$ represents the exclusive choice between children, e.g., $\phi(\times(a, b)) = \{\langle a \rangle, \langle b \rangle\}$
- $\wedge$ denotes the concurrent composition of children, e.g., $\phi(\wedge(a, b)) = \{\langle a, b \rangle, \langle b, a \rangle\}$
- $\circlearrowleft$ represents the loop execution in which the first child is the body of the loop and the other children are redo children, e.g., $\phi(\circlearrowleft (a, b)) = \{\langle a \rangle, \langle a, b, a \rangle, \dots\}$.

$M = \times(\rightarrow (a, b), \wedge(\times(c, \tau), d))$ is a more complex example such that $\phi(M) = \{\langle a, b \rangle, \langle d \rangle, \langle c, d \rangle, \langle d, c \rangle\}$.

Given a set of activities $\Sigma \subseteq \mathcal{A}$, among all possible process tree models that can be generated using Σ, there exists a specific model $M \in \mathcal{M}_\Sigma$ that allows for all possible sequences of activities, i.e., $\phi(M) = \Sigma^*$. This model is referred to as the *flower model*. For instance, for $\Sigma = \{a, b\}$, the process tree $\circlearrowleft (\tau, a, b)$ represents the corresponding flower model, as it permits all permutations and repetitions of the activities in Σ.

Definition 4 (*Flower Model*). Let $\Sigma = \{a_1, \dots, a_n\} \subseteq \mathcal{A}$ be a subset of activities. The flower model is a process tree that allows for all possible combinations of the nodes, i.e., $\circlearrowleft (\tau, a_1, \dots a_n)$. $\mathfrak{flo}(\Sigma)$ represent the flower tree generated over the set of activities Σ. The language of this model is $\phi(\mathfrak{flo}(\Sigma)) = \Sigma^*$.

Directly-Follows Graphs (DFGs) are an intermediate process representation commonly used in inductive mining techniques to identify suitable process structures during model discovery.

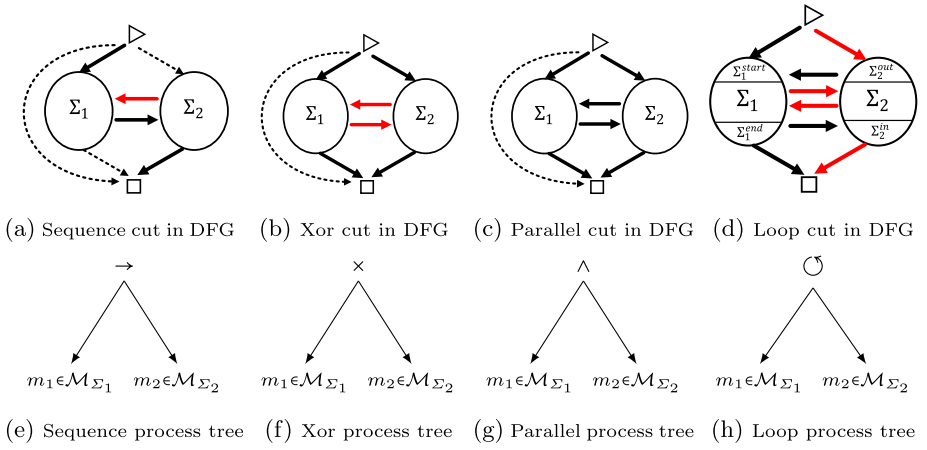


Fig. 1. The correspondence between the binary cuts and the process tree structures.

Definition 5 (*Directly-Follows Graph*)**.** Let $L \in \mathcal{L}$ be an event log. The DFG of this event log denoted as $\mathcal{G}(L) = (\Sigma, E)$, is constructed, where:

- $\Sigma = \{a \mid a \in \sigma, \sigma \in L\}$ represents the set of activities, and
- $E = [(\sigma(i), \sigma(i+1)) \mid \sigma \in L', 1 \leq i < |\sigma|]$, with $L' = [\langle \triangleright \rangle . \sigma . \langle \square \rangle \mid \sigma \in L]$, denotes the multiset of edges capturing directly-follows relationships. Here, artificial start ($\triangleright$) and end ($\square$) nodes are added, ensuring completeness in the graph representation.

4. Inductive Miner with rules (IMr)

The original inductive mining technique for process model discovery, introduced in [25], and its subsequent adaptations [20,26], established a crucial line of research that has proven to be among the most promising approaches in process discovery. These techniques incorporate the notion of *fall-throughs*, i.e., fallback mechanisms that ensure a process model can still be constructed when no process structure is found that reflects no deviation or missing behavior when compared with the given event log. To address the limitations of fall-throughs, the IMbi algorithm proposed in [8] extends the inductive mining framework by introducing cost functions that quantify the quality of possible process structures at each recursion. This allows the algorithm to select the most promising process structure, even when a perfect one is not available, resulting in process models that are more frequency-aware and less prone to overgeneralization.

The central idea of the IMbi framework, which is adapted in the IMr framework as well, is to create a DFG $\mathcal{G}(L) = (\Sigma, E)$ considering the event log L and recursively partition the activity set Σ into two subsets, guided by an operator $\oplus$. These operators correspond to process tree composition operators as defined in Definition 3, namely $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$.

Definition 6 (*Binary Cut*)**.** Let $L \in \mathcal{L}$ be an event log. $\mathcal{G}(L) = (\Sigma, E)$ is the corresponding DFG. A binary cut $(\oplus, \Sigma_1, \Sigma_2)$ divides Σ into two partitions, such that $\Sigma_1 \cup \Sigma_2 = \Sigma$, $\Sigma_1 \cap \Sigma_2 = \emptyset$, and $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$ is a cut type operator. C_Σ is the universe of binary cuts we can generate with the set of nodes Σ.

In Fig. 1, the correspondence between the binary cuts and the process tree structures is represented. In the DFGs, the red lines are the deviating edges, the dashed lines are the optional edges, and the solid lines are the mandatory edges according to the semantics of the process trees corresponding to them. For example, in the sequence cut, all the edges should be from Σ_1 to σ_2. Any edge from Σ_2 to Σ_1 is considered deviating. Considering the set of nodes Σ there are potentially a large number of possibilities to divide the set of activities into two sets with cut operators.

The IMbi framework adopts a divide-and-conquer approach for process discovery, recursively discovering process structures that support a desirable event log while avoiding an undesirable event log. The algorithm starts by taking as input:

- a desirable event log $L^+ \in \mathcal{L}$,
- an undesirable event log $L^- \in \mathcal{L}$,
- $ratio \in [0, 1]$ parameter that controls the effectiveness of the undesirable event log, and
- $sup \in [0, 1]$ parameter integrated to balance the importance of missing behavior.

Considering $cost^{\oplus}_{\mathcal{G}(L)} : C_\Sigma \times [0,1] \rightarrow \mathbb{R}$ as a cost function that assigns cost to each candidate cut $c \in C_\Sigma$ considering the $sup \in [0, 1]$ parameter and $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$, the main objective function in each recursion is:

$$ov\_cost_{G^+,G^-}(c, sup, ratio) = cost^{\oplus}_{G^+}(c, sup) - ratio \cdot cost^{\oplus}_{G^-}(c, sup)$$

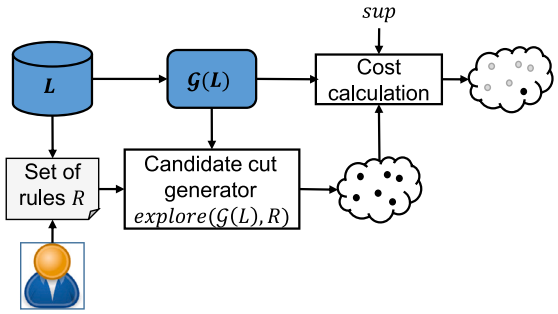


Fig. 2. One recursion of IMr, the framework proposed in this paper, identifies candidate cuts adhering to specified rules and selects the cut with minimum cost, incorporating cost functions from [8].

The best cut in each recursion is the cut with the minimum ov_cost, i.e.,

$$optimal\_cut = \arg\min_{c \in C_\Sigma} \{ov\_cost_{G^+,G^-}(c, sup, ratio)\}$$

The IMr technique introduced in [1] and extended in this paper makes the following adaptations on the IMbi technique:

- Selection of $ratio = 0$ ignores the undesirable event log and regulates the focus of process discovery to one event log, i.e., the desirable event log L^+. However, since only one event log is used, we refer to it as L.
- A set of rules, denoted by R, is considered as an additional input besides the input event log. These rules are instantiations of predefined rule templates using activities from the event log. The rule set R is then used to prune the set of candidate cuts C_Σ by eliminating process structures that generate behaviors deviating from these rules.

The key concepts of the IMr framework are summarized in Fig. 2 and an overview of its recursive algorithmic workflow is presented in Algorithm 1.

Algorithm 1 IMr Framework

1: **function** $disc(L, sup, R)$
2: ▷ *$L \in \mathcal{L}$ is an event log, $sup \in [0, 1]$ is a process discovery parameter and R is the set of rules.* ◁
3: $base = checkBaseCase(L, sup)$
4: **if** $checkBaseCase$ successful **then**
5: **return** $base$
6: $C = explore(\mathcal{G}(L), R)$
7: ▷ *The function $explore(\mathcal{G}(L), R)$ is one of the main contributions of the paper and is explained in Definition 13.* ◁
8: $(\oplus, \Sigma_1, \Sigma_2) = \arg\min_{c \in C} \{ov\_cost_{\mathcal{G}(L)}(c, sup)\}$
9: $L_1, L_2 = SPLIT(L, (\oplus, \Sigma_1, \Sigma_2))$
10: **return** $\oplus(disc(L_1, sup, R), disc(L_2, sup, R))$

At each recursion step, the algorithm performs the following tasks:

(1) **Base Case Check:** Verifies whether the event log consists exclusively of traces with a single activity or empty traces (lines 3–5 in Algorithm 1). If this condition is met, the recursion terminates.
(2) **Selection of Binary Cuts:** Among the possible binary cuts $explore(\mathcal{G}(L), R)$ explore the directly follows graph $\mathcal{G}(L)$ and identifies the candidate cuts with the minimum deviation from the input rules (lines 6 in Algorithm 1). This step is a key contribution of this paper and is explained in detail in the following sections.
(3) **Cut Evaluation and Selection:** Computes the cost for each candidate cut by assessing the number of deviating edges and estimating the missing edges required to align $\mathcal{G}(L)$ with the candidate cut [8]. Based on this cost function, the algorithm selects the most suitable binary cut, considering the observed behavior in L and the sup parameter (line 7 in Algorithm 1).
(4) **Event Log Splitting:** Splits the event log L into subsets corresponding to the selected cut and proceeds to the next recursion (lines 8–9 in Algorithm 1).

The recursive application of this divide-and-conquer strategy leads to the hierarchical decomposition of the process structure, allowing for the discovery of increasingly complex branching behaviors.

5. Foundations and strategies for rule-guided process discovery

This section presents a comprehensive approach to incorporating rules into process discovery. Rules serve as constraints to guide the discovery process, ensuring that the resulting models adhere to predefined requirements or expectations. To effectively integrate rules, it is necessary to define their structure, formalize their semantics, and validate their compliance with the discovered models. Additionally, challenges such as rule conflicts and deviations must be addressed systematically. This section provides the necessary foundations for rule formalization, validation at both the model and intermediate recursion levels, and the quantification of rule deviations in discovered models.

The input set of rules can originate from various sources but must be formalized in a manner compatible with the IMr framework. The IMr framework, as introduced in [1], presents several limitations that we address in this section by providing a mathematical foundation:

- The decision-making process regarding which cuts should be excluded based on a given rule is currently ad hoc and handled on a case-by-case basis. No formal mechanism for automatic verification is provided. In this paper, we address this limitation by introducing a formalized approach grounded in mathematical reasoning.
- A specific situation may arise when all available cuts deviate at least one of the given rules as input. The original framework handles this by ignoring the rules and selecting the best cut without them. We improve upon this by introducing a metric that ranks candidate cuts based on the severity of rule violations, enabling a smoother decision process that avoids a binary include-or-ignore approach.
- No metric is provided in the original framework to assess whether the discovered models conform to the given rules. We address this gap by introducing quantitative metrics to evaluate the extent to which rule adherence has been achieved during the discovery process.

5.1. Defining and formalizing rules

Different rule specification languages may require specific formalizations to ensure their effective integration. As a general guideline, we first define the concept of a rule within the context of our framework in a broad, abstract manner. Subsequently, we refine this definition to align with a specific rule specification language, namely declarative constraints.

Definition 7 (*Rules*)**.** Let $\mathcal{T}$ be a non-empty set of templates, where each template is a relation $d(x_1, \dots, x_m) \in \mathcal{T}$ defined over variables $x_1, \dots, x_m$, with $m \in \mathbb{N}$ representing the arity of d. For a given set of activities $a_1, \dots, a_m \in \mathcal{A}$, a rule $d(a_1, \dots, a_m)$ is derived as a specific instantiation of the template d, binding its variables to the corresponding activities. $\mathcal{R}$ is the universe of rules.

The template collection $\mathcal{T}$ can be customized according to design preferences. To evaluate a candidate cut, it is necessary for the rules to adhere to a specific formal language. This ensures that each trace generated over the universe of activities is either accepted or rejected by the rule.

Definition 8 (*Language of a Rule*)**.** Let $r \in \mathcal{R}$ be a rule. Any trace $\sigma \in \mathcal{A}^*$ is allowed by r unless there is explicit evidence that the trace violates the constraint, as determined by the semantics of the associated template. Such a violation is denoted as $\sigma \not\models r$. The language of a rule, denoted as $\phi(r)$, is the set of all traces that satisfy the constraint, i.e., $\phi(r) = \{\sigma \in \mathcal{A}^* \mid \sigma \models r\}$.

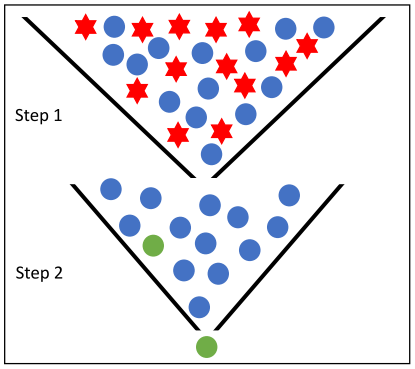
For instance, $Response(x_1, x_2)$ can serve as a template representing the constraint that if x_1 occurs, then x_2 must follow at some point afterward. Given a set of activities $\{a, b, c, d\}$, $Response(a, b)$ is a specific instantiation of this template. The trace $\langle a, b, c\rangle$ satisfies $Response(a, b)$ ($\langle a, b, c\rangle \models Response(a, b)$) because b occurs after a. Conversely, the trace $\langle c, d, a\rangle$ does not satisfy $Response(a, b)$ ($\langle c, d, a\rangle \not\models Response(a, b)$) because the occurrence of a at the end of the trace is not followed by b.

5.2. Rule validation in process tree models

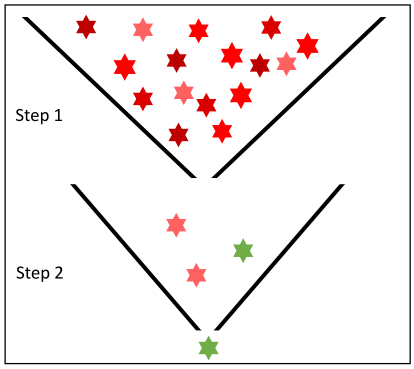
The final models discovered by the IMr algorithm are process trees. From the discovered process tree, we can derive its language and compare it against the given set of rules. Both the process model and each rule classify traces into two categories: accepted or rejected. Consider the process tree $M = \rightarrow(\rightarrow(a, c), \times(b, d))$. For any trace $\sigma \in \{a, b, c, d\}^*$, one of the following cases will hold true:

- $\sigma \in \phi(M)$ and $\sigma \in \phi(r)$: The trace is accepted by both the model and the rule (e.g., $\langle a, c, b\rangle$).
- $\sigma \in \phi(M)$ and $\sigma \notin \phi(r)$: The trace is accepted by the model but rejected by the rule (e.g., $\langle a, c, d\rangle$).
- $\sigma \notin \phi(M)$ and $\sigma \in \phi(r)$: The trace is rejected by the model but accepted by the rule (e.g., $\langle b, a, b\rangle$).
- $\sigma \notin \phi(M)$ and $\sigma \notin \phi(r)$: The trace is rejected by both the model and the rule (e.g., $\langle d, a\rangle$).

Definition 9 (*Rule Violation in Process Trees*)**.** Let $M \in \mathcal{M}$ be a process tree and $r \in \mathcal{R}$ be a rule. The model M violates the rule r, denoted as $M \not\models r$, if there exists a trace $\sigma \in \phi(M)$ such that $\sigma \not\models r$. In other words, the language of the process tree M contains at least one trace that does not satisfy the rule r. Conversely, if no such trace exists, the model M satisfies the rule r, denoted as $M \models r$.



(a) Candidate cuts categorized based on rule violations. Red stars represent cuts violating some rules from R, and blue circles represent cuts violating no rules. The green circle is the cut with the minimum cost.



(b) Ranking candidate cuts when all violate at least one rule. Cuts with less severe deviations are chosen in the candidate cut selection step and among them, the one with the minimum cost is selected, represented as green star.

Fig. 3. Illustration of the two-step approach for selecting the best cut. In the first step, among all possible candidate cuts, the cuts with less rule deviations are selected. The cut with the minimum cost based on the event log is selected in the second step.

5.3. Rule validation in binary cuts

To influence the discovery algorithm, it is essential to intervene during the intermediate recursion steps, where the final model has not yet been constructed, and multiple possibilities exist for continuing the model construction. At these intermediate stages, the algorithm operates on an event log as input, along with a set of rules. It explores possible cuts, ranks them using cost functions, and selects the cut with the minimum cost. The rules play a crucial role during the candidate cut generation step, where numerous potential cuts may exist. Some of these candidate cuts lead to models that allow traces that should not be permitted in the final model. To address this, the algorithm aims to eliminate such candidate cuts by leveraging the given rules.

Consider a candidate cut $c = (\oplus, \Sigma_1, \Sigma_2)$ during an intermediate recursion. Selecting this cut may result in multiple possible process trees being generated in subsequent steps. The objective is to guide the selection process to ensure that the generated trees adhere as closely as possible to the rules, thereby improving the alignment between the discovered model and the input constraints.

Definition 10 (*Model Collection of a Binary Cut*)**.** Let $L \in \mathcal{L}$ be an event log, $\mathcal{G}(L) = (\Sigma, E)$ be the extracted DFG, and $c = (\oplus, \Sigma_1, \Sigma_2) \in C_\Sigma$ be a binary cut, $\mathcal{M}_c = \{\oplus(M_1, M_2) | M_1 \in \mathcal{M}_{\Sigma_1} \wedge M_2 \in \mathcal{M}_{\Sigma_2}\}$ is the universe of process tree models we can generate by selecting the cut c.

If all the models in $\mathcal{M}_c$ violate the rule r, it can be concluded that the final model, irrespective of the subsequent recursion steps, will also violate this rule.

Definition 11 (*Rule Violation in Binary Cuts*)**.** Let $L \in \mathcal{L}$ be an event log, $\mathcal{G}(L) = (\Sigma, E)$ be the DFG extracted from L, and $R \subseteq \mathcal{R}$ be a set of rules. A cut $c \in C_\Sigma$ is said to violate a constraint $r \in R$, denoted as $c \not\models r$, if for all process trees $M \in \mathcal{M}_c$, there exists a trace $\sigma \in \phi(M)$ such that $\sigma \not\models r$.

Our strategy works as follows: First, we remove the binary cuts that violate some rules from the given input rule set R and then allow the event log to choose a suitable structure and recursively continue until we have the final model. Among all possible candidate cuts, we select a subset that aligns most closely with the given input rules. In Fig. 3(a), the red stars denote candidate cuts that violate one or more rules from R, whereas the blue circles represent candidate cuts that conform to all the rules in R. Ideally, at this stage, we aim to eliminate cuts that conflict with the rules entirely. However, when all candidate cuts deviate from at least one rule, we prioritize them based on the severity of their deviations and select the cut with the least severe violation, as depicted in Fig. 3(b). Specifically, the cost associated with a violated rule depends on its criticality: cuts that violate highly significant rules incur higher costs compared to those that deviate from less critical rules, such as those satisfied in only a few traces.

5.4. Challenges and solutions in rule-based candidate selection

This strategy has two potential limitations:

- Even if all rules are satisfied during every recursion step, there is no guarantee that the final model will satisfy these rules.
- The use of rules to prune the set of candidate cuts may lead to an empty set if all candidates conflict with at least one rule $r \in R$.

The first limitation arises from the representational bias of the algorithm and is inherently unavoidable. Due to the recursive nature of the framework, the discovery task is progressively divided into smaller subtasks during each recursion. This process continues until all subbranches are resolved, after which the results are combined. Dividing the discovery task can cause activities involved in a rule to end up in different branches, making it impossible to evaluate the rule in subsequent steps. For instance, consider the rule $Response(a, b)$ and the cut $(\rightarrow, \{a, c\}, \{b, d\})$. After this cut, a and c are processed in one branch, while b and d are in another. Consequently, the rule $Response(a, b)$ can no longer be evaluated in subsequent steps, and the model construction process might generate a structure that violates this rule.

To address the second limitation, when no cut satisfies all the rules without deviations, we select the most promising cuts based on their importance, optionally provided as input. To implement this, we assign an importance weight to each input rule $r \in R$. Candidate cuts with the lowest deviations are allowed to remain in the candidate set, even if some input rules are expected to be violated in the final model. Alternatively, importance weights can be determined using domain knowledge or automated methods, such as calculating the *support* or *confidence* of the rules based on the event log.

Definition 12 (*Deviation Penalty*)**.** Let $R \subseteq \mathcal{R}$ be a set of rules provided by the user or automatically discovered. The function $dev\text{-}penalty_R : R \rightarrow [0, 1]$ assigns a penalty to the violation of each rule, reflecting the rule's importance.

The penalty value assigned to a rule by $dev\text{-}penalty_R$ represents its significance. If no deviation penalty function is specified by the user, we assume equal $dev\text{-}penalty_R$ values for all rules.

Definition 13 (*The Set of Candidate Cuts*)**.** Let $L \in \mathcal{L}$ be an event log and $R \subseteq \mathcal{R}$ be a set of rules. $\mathcal{G}(L) = (\Sigma, E)$ is the corresponding DFG. Considering the $dev\text{-}penalty_R$ as the function that assigns a penalty to the violation of each rule in R. We define the minimum penalty as:

$$minimum\text{-}penalty = \min_{c \in C_\Sigma} \left\{ \sum_{r \in R | c \not\models r} dev\text{-}penalty_R(r) \right\}$$

The function $explore(\mathcal{G}(L), R)$ identifies the candidate cuts with the minimum penalty amount.

$$explore(\mathcal{G}(L), R) = \{c \in C_\Sigma \mid \sum_{r \in R | c \not\models r} dev\text{-}penalty_R(r) = minimum\text{-}penalty\}$$

If there are cuts with no deviations, $explore(\mathcal{G}(L), R)$ includes all such cuts. However, if no deviation-free cut exists, $explore(\mathcal{G}(L), R)$ contains the subset of cuts with the lowest total deviation penalty.

5.5. Metrics for quantifying rule deviations

Since the satisfaction of all rules in the end is not guaranteed, we define metrics to quantify the severity of the deviation.

Definition 14 (*Deviation Count and Deviation Rate*)**.** Let $R \subseteq \mathcal{R}$ be a set of rules, and $M \in \mathcal{M}$ be a process tree model. The function $dev\text{-}count : \mathcal{M} \times \mathcal{R} \rightarrow \mathbb{N}$ calculates the number of rules violated by the given process model M, defined as:

$$dev\text{-}count(M, R) = |\{r \in R \mid M \not\models r\}|.$$

The function $dev\text{-}rate : \mathcal{M} \times \mathcal{R} \rightarrow [0, 1]$ computes the normalized deviation rate by summing the deviation penalties of the violated rules and dividing by the total penalty of all rules. It is defined as:

$$dev\text{-}rate(M, R) = \frac{\sum_{r \in R | M \not\models r} dev\text{-}penalty_R(r)}{\sum_{r \in R} dev\text{-}penalty_R(r)}.$$

6. Declarative constraints and computational feasibility in rule-based discovery

Verifying whether, for all models $M \in \mathcal{M}_c$, and for all rules $r \in R$, $M \models r$ is computationally expensive. Performing such calculations across all recursions renders the solution computationally infeasible and non-scalable. In this section, we focus on declarative constraints as rules and discuss strategies to address the computational cost challenge.

6.1. Declarative constraints as rules

Declarative constraints serve as a flexible and interpretable approach for modeling control-flow behaviors in processes. By leveraging generic templates to define behavioral rules and assigning specific activity labels, they enable a rich and expressive representation of constraints that can be systematically assessed using event logs. The declarative templates used in this paper capture a range of control-flow behaviors. These templates are as follows:

- $AtLeast1(x_1)$: x_1 occurs at least once.
- $AtMost1(x_1)$: x_1 occurs at most once.
- $Absence(x_1)$: x_1 does not occur.

- $Init(x_1)$: x_1 is the first to occur.
- $End(x_1)$: x_1 is the last to occur.
- $RespondedExistence(x_1, x_2)$: If x_1 occurs, x_2 occurs as well.
- $CoExistence(x_1, x_2)$: If x_1 occurs, x_2 occurs as well and vice versa.
- $Response(x_1, x_2)$: If x_1 occurs, then x_2 occurs after x_1,
- $AlternateResponse(x_1, x_2)$: If x_1 occurs, then x_2 occurs afterwards before x_1 recurs.
- $ChainResponse(x_1, x_2)$: If x_1 occurs, then x_2 occurs immediately after it.
- $Precedence(x_1, x_2)$: x_2 occurs only if preceded by x_1,
- $AlternatePrecedence(x_1, x_2)$: x_2 occurs only if preceded by x_1 with no other x_2 in between.
- $ChainPrecedence(x_1, x_2)$: x_2 occurs only if x_1 occurs immediately before it.
- $Succession(x_1, x_2)$: x_1 occurs if and only if it is followed by x_2.
- $AlternateSuccession(x_1, x_2)$: x_1 and x_2 occur if and only if they follow one another, alternating.
- $ChainSuccession(x_1, x_2)$: x_1 and x_2 occurs if and only if x_2 immediately follows x_1.
- $NotRespondedExistence(x_1, x_2)$: If x_1 occurs, then x_2 does not occur.
- $NotCoExistence(x_1, x_2)$: x_1 and x_2 never occur together
- $NotResponse(x_1, x_2)$: If x_1 occurs, then x_2 does not occur after x_1.
- $NotChainResponse(x_1, x_2)$: If x_1 occurs, then x_2 does not occur immediately after x_1.
- $NotPrecedence(x_1, x_2)$: x_2 occurs only if it is not preceded by x_1.
- $NotChainPrecedence(x_1, x_2)$: x_2 occurs only if x_1 does not occur immediately before it.
- $NotSuccession(x_1, x_2)$: x_2 cannot occur after x_1.
- $NotChainSuccession(x_1, x_2)$: x_1 and x_2 must not appear consecutively.

This set of 24 declarative constraint templates is commonly used in the declarative process mining literature [27] and supported by tools such as MINERful [28] and RUM [29]. Among these templates, the IMr framework proposed in [1] utilizes a selected subset, specifically eight templates: $AtMost1(x_1)$, $AtLeast1(x_1)$, $RespondedExistence(x_1, x_2)$, $CoExistence(x_1, x_2)$, $Response(x_1, x_2)$, $Precedence(x_1, x_2)$, $NotCoExistence(x_1, x_2)$, and $NotSuccession(x_1, x_2)$[1].

Considering this set of templates as $\mathcal{T}_{decl}$, we can generate rules according to Definition 7, referring to them as $\mathcal{R}_{decl} \subseteq \mathcal{R}$. Process experts can capture their knowledge and insights as declarative rules, which can then be utilized in process discovery, as discussed in this paper. Beyond user-defined rules, automated declarative process discovery tools, such as Declare Miner [16] and MINERful [17], offer the capability to extract declarative constraints directly from event logs.

Declarative constraints can be assessed using quality measures such as *confidence* and *support*, which provide insights into their applicability and effectiveness [27]. These measures are evaluated at two distinct levels: the *trace level*, which considers entire process instances, and the *event level*, which focuses on individual occurrences of activities. For example, in the trace $\langle c, b, a, b, b, c \rangle$, the constraint $Precedence(a, b)$ is violated at the trace level because the first occurrence of b is not preceded by a. However, at the event level, b appears three times, with the constraint satisfied for two occurrences and violated for one. Event-level analysis offers a more granular perspective, ensuring that instances of constraint satisfaction are not overlooked. In this paper, we adopt the event-level approach to achieve a detailed and nuanced evaluation of constraint quality.

Each declarative constraint is associated with an *activation*, which defines the conditions under which the constraint is evaluated. For instance, in the constraints $Response(a, b)$ and $Precedence(a, b)$, the activations are triggered by a and b, respectively. If no activation occurs within a trace, there is no basis to determine whether the constraint is satisfied or violated; in such cases, the constraint is considered *vacuously satisfied* [27]. While vacuous satisfaction is a natural occurrence, relying on it exclusively can lead to misleading conclusions. To mitigate this, we focus solely on instances where constraints are *non-vacuously satisfied*, ensuring that evaluations are grounded in meaningful activations.

Definition 15 (*Support and Confidence*). Let $\mathcal{R}_{decl} \subseteq \mathcal{R}$ be the universe of rules we can defined with the declarative constraints templates. Let $L \in \mathcal{L}$ be an event log. For a given declarative constraint $r \in \mathcal{R}_{decl}$, let the following be defined:

- $\#_{activated}(L, r)$: The number of times the constraint r is activated in L.
- $\#_{satisfied}(L, r)$: The number of times the constraint r is satisfied in L.
- $\#_{events}(L)$: The total number of events in L.

Using these quantities, the coverage and confidence of a constraint r are defined as follows:

- $Support : \mathcal{L} \times \mathcal{R}_{decl} \to [0, 1]$ is the proportion of activated events to the total number of events in L:

$$Support(L, r) = \frac{\#_{satisfied}(L, r)}{\#_{events}(L)}.$$

[1] The constraints $AtLeast1$ and $AtMost1$ are referred to as *existence* and *at-most* constraints, respectively, in [1].

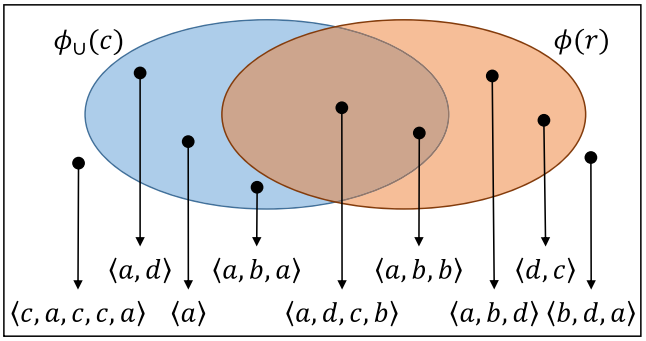


Fig. 4. Example traces illustrating the relationships between $\phi_{\cup}(c)$ and $\phi(r)$ for the cut $c = (\rightarrow, \{a, d\}, \{c, b\})$ and the rule $r = Response(a, b)$.

– $Confidence : \mathcal{L} \times \mathcal{R}_{decl} \rightarrow [0, 1]$ is the proportion of satisfied events to the number of activated events in L. To avoid division by zero, the denominator is set to at least 1:

$$Confidence(L, r) = \frac{\#_{satisfied}(L, r)}{\max(1, \#_{activated}(L, r))}.$$

6.2. Evaluating rule compliance in candidate cuts

Next, we explore the relationship between the languages of candidate binary cuts and their compliance with input rules. By examining the union of all possible model languages generated by a candidate cut, we gain insights into which traces can or cannot be produced by these models. This analysis provides a foundation for evaluating whether a cut aligns with or violates the given rules, guiding the discovery algorithm towards optimal choices. Additionally, we define common modes, i.e., subsets of traces shared across all models generated by all potential models of a binary cut, and demonstrate how they can be used to identify rule violations effectively. Through this framework, we formalize the evaluation of cuts and establish criteria for their selection in rule-informed process discovery.

Union of model languages and rule violations

For each binary cut, there may be numerous process tree models that can be generated in subsequent recursion steps. Each time a cut is selected during recursion, the possible language of the final model becomes more constrained. Although the exact process model and its corresponding language are not determined at intermediate recursion steps, it is possible to define an upper bound for the language allowed by the selected cut.

Definition 16 (*Union Language of a Cut*)**.** Let $L \in \mathcal{L}$ be an event log, and $\mathcal{G}(L) = (\Sigma, E)$ its corresponding DFG. For a binary cut $c \in C_{\Sigma}$, the union language of c is defined as the union of the languages of all process tree models that can be generated using this cut: $\phi_{\cup}(c) = \bigcup_{M \in \mathcal{M}_c} \phi(M)$.

If a trace belongs to $\phi_{\cup}(c)$, it indicates that at least one model in $\mathcal{M}_c$ can generate this trace. Conversely, if a trace does not belong to $\phi_{\cup}(c)$, no model in $\mathcal{M}_c$ can generate it. It is important to note, considering a rule $r \in R$,

– A model at the end *might violate* the input rules if $\exists_{M \in \mathcal{M}_c} M \not\models r$
– A model at the end *violates* the input rules if $\forall_{M \in \mathcal{M}_c} M \not\models r$

For example, consider $\Sigma' = \{a, b, c, d\}$, a candidate cut $c = (\rightarrow, \{a, d\}, \{c, b\})$, and the rule $r = Response(a, b)$. Fig. 4 illustrates example traces and their relationships with $\phi_{\cup}(c)$ and $\phi(r)$:

– $\sigma \notin \phi_{\cup}(c)$, $\sigma \notin \phi(r)$: Traces such as $\langle c, a, c, c, a\rangle$ and $\langle b, d, a\rangle$ are not allowed by any model $M \in \mathcal{M}_c$ nor by the rule $r = Response(a, b)$.
– $\sigma \in \phi_{\cup}(c)$, $\sigma \notin \phi(r)$: Traces such as $\langle a, d\rangle$, $\langle a\rangle$, and $\langle a, b, a\rangle$ belong to the union of the models but are not permitted by the rule r.
– $\sigma \in \phi_{\cup}(c)$, $\sigma \in \phi(r)$: Traces such as $\langle a, d, c, b\rangle$ and $\langle a, b, b\rangle$ are allowed by both the union of the models and the rule r.
– $\sigma \notin \phi_{\cup}(c)$, $\sigma \in \phi(r)$: Traces such as $\langle a, b, d\rangle$ and $\langle d, c\rangle$ are permitted by the rule r but cannot be generated by any model in $\mathcal{M}_c$.

Since Σ_1^* and Σ_2^* allow for any behavior in the resulting branches of the process tree, it follows that $\phi_{\cup}(c) \subseteq \oplus_{\phi}(\Sigma_1^*, \Sigma_2^*)$. Moreover, the flower model is always a possible model in the next recursion steps; that is, $\mathfrak{fl}(\Sigma_1) \in \mathcal{M}_{\Sigma_1}$ and $\mathfrak{fl}(\Sigma_2) \in \mathcal{M}_{\Sigma_2}$. Consequently, the model $\oplus(\mathfrak{fl}(\Sigma_1), \mathfrak{fl}(\Sigma_2))$ is a valid candidate within $\mathcal{M}_c$, with its language being $\oplus_{\phi}(\Sigma_1^*, \Sigma_2^*)$. Therefore, $\phi_{\cup}((\oplus, \Sigma_1, \Sigma_2)) = \oplus_{\phi}(\Sigma_1^*, \Sigma_2^*)$.

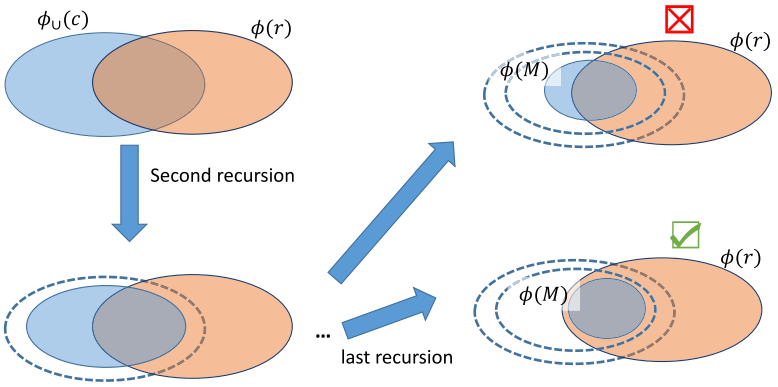


Fig. 5. Recursive restriction of the allowed language in the IMr algorithm, starting with Σ^* and progressively narrowing to $\oplus_\phi(\Sigma_1^*, \Sigma_2^*)$ at each step, until the final model M is constructed.

Lemma 1. *Let $L \in \mathcal{L}$ be an event log. $\mathcal{G}(L) = (\Sigma, E)$ is the corresponding DFG. Considering $(\oplus, \Sigma_1, \Sigma_2) \in C_\Sigma$ as a binary cut, we can conclude that: $\phi_\cup((\oplus, \Sigma_1, \Sigma_2)) = \oplus_\phi(\Sigma_1^*, \Sigma_2^*)$.*

Proof. We $\phi_\cup(c) \subseteq \oplus_\phi(\Sigma_1^*, \Sigma_2^*)$ and $\oplus_\phi(\Sigma_1^*, \Sigma_2^*) \subseteq \phi_\cup(c)$ and then conclude $\phi_\cup(c) = \oplus_\phi(\Sigma_1^*, \Sigma_2^*)$

(1) $\boldsymbol{\phi_\cup(c) \subseteq \oplus_\phi(\Sigma_1^*, \Sigma_2^*)}$. By Definition 16, $\phi_\cup(c) = \bigcup_{M \in \mathcal{M}_c} \phi(M)$, where $\mathcal{M}_c = \{\oplus(M_1, M_2) \mid M_1 \in \mathcal{M}_{\Sigma_1}, M_2 \in \mathcal{M}_{\Sigma_2}\}$. For any $M_1 \in \mathcal{M}_{\Sigma_1}$, we have $\phi(M_1) \subseteq \Sigma_1^*$, and similarly $\phi(M_2) \subseteq \Sigma_2^*$. By monotonicity of the language-join $\oplus_\phi$ (i.e. larger operand languages yield larger joined languages),

$$\forall M_1 \in \mathcal{M}_{\Sigma_1} \wedge \forall M_2 \in \mathcal{M}_{\Sigma_2} : \quad \phi\big(\oplus(M_1, M_2)\big) = \oplus_\phi\big(\phi(M_1), \phi(M_2)\big) \subseteq \oplus_\phi(\Sigma_1^*, \Sigma_2^*).$$

Taking the union over all M_1, M_2 thus gives $\phi_\cup(c) = \bigcup_{M \in \mathcal{M}_c} \phi(M) \subseteq \oplus_\phi(\Sigma_1^*, \Sigma_2^*)$.

(2) $\boldsymbol{\oplus_\phi(\Sigma_1^*, \Sigma_2^*) \subseteq \phi_\cup(c)}$. By the definition of the flower model (Definition 4), for each $i \in \{1, 2\}$ the model $\mathfrak{fl}(\Sigma_i)$ satisfies $\phi\big(\mathfrak{fl}(\Sigma_i)\big) = \Sigma_i^*$ and $\mathfrak{fl}(\Sigma_i) \in \mathcal{M}_{\Sigma_i}$. Hence, the process tree $\oplus\big(\mathfrak{fl}(\Sigma_1), \mathfrak{fl}(\Sigma_2)\big)$ is a valid element of $\mathcal{M}_c$, and its language is

$$\phi\big(\oplus(\mathfrak{fl}(\Sigma_1), \mathfrak{fl}(\Sigma_2))\big) = \oplus_\phi\big(\phi(\mathfrak{fl}(\Sigma_1)), \phi(\mathfrak{fl}(\Sigma_2))\big) = \oplus_\phi(\Sigma_1^*, \Sigma_2^*).$$

Therefore, $\oplus_\phi(\Sigma_1^*, \Sigma_2^*)$ is one of the languages in the union defining $\phi_\cup(c)$, giving $\oplus_\phi(\Sigma_1^*, \Sigma_2^*) \subseteq \phi_\cup(c)$.

Conclusion. Combining (1) and (2) yields $\phi_\cup(c) = \oplus_\phi(\Sigma_1^*, \Sigma_2^*)$.

With each recursion in the IMr algorithm, selecting a cut progressively restricts the language allowed by the final model. Initially, all behavior is permitted, i.e., Σ^*. After the first cut, the allowed behavior becomes more constrained, limited to at most $\oplus_\phi(\Sigma_1^*, \Sigma_2^*)$. This process continues recursively until the final model is constructed. In Fig. 5, let $\phi(r)$ denote the language of a specific rule $r \in \mathcal{R}$. At each recursion step, the algorithm selects a cut, further narrowing the set of allowed traces. The recursion proceeds until the final step, where the final model M is discovered. Ideally, the language of the final model should be a subset of the language of the rule r. This implies that all traces allowed by M are also permitted by r, i.e., $M \vDash r$.

If trace $\sigma \in \phi_\cup(c)$, some possible models might allow for it and some might not. For example, the process tree $M_1 = \rightarrow(\rightarrow(a, d), \rightarrow(c, b))$ allow for $\langle a, d, c, b\rangle$ but $M_2 = \rightarrow(\times(a, d), \times(c, b))$ does not allow for it. In this example, $\phi(M_1) = \{\langle a, d, c, b\rangle\}$, and $\phi(M_2) = \{\langle a, c\rangle, \langle a, b\rangle, \langle d, c\rangle, \langle d, b\rangle\}$. $M_1 \vDash r$ because there is no trace in the language of the model that violates the rule. $M_2 \nvDash r$ because trace $\langle a, c\rangle$ violates r.

As discussed in Section 5.4, the final model cannot be guaranteed to fully satisfy all rules due to the representational bias of the algorithm and the potential necessity of selecting violating cuts when no non-violating cuts exist. Nevertheless, the objective remains to guide the discovery process and maximize the influence of the input rules.

Common modes in candidate cuts

Considering a set of activities $\Sigma \subseteq \mathcal{A}$, Σ^* denotes the set of all possible traces over Σ. The set $\mathcal{M}_\Sigma$ contains all possible process tree models that can be generated using Σ. For any activity $a \in \Sigma$, let $\Sigma_{\setminus a}$ denote the set of activities excluding a, i.e., $\Sigma_{\setminus a} = \Sigma \setminus \{a\}$. The following properties hold:

- Traces without a: The set $\Sigma_{\setminus a}^*$ contains all traces over Σ where activity a does not occur. The model $\rightarrow(a, \mathfrak{fl}(\Sigma_{\setminus a})) \in \mathcal{M}_\Sigma$ is an example of a possible model that ensures activity a cannot be skipped, and therefore, $\exists_{M \in \mathcal{M}_\Sigma} \Sigma_{\setminus a}^* \cap \phi(M) = \emptyset$.

- Traces with two or more occurrences of a: The set $(\Sigma^*_{\setminus a} \circledast \boldsymbol{a}\{2,\})$ contains all traces where a occurs at least twice. The model $\rightarrow(a, \hat{\mathfrak{f}}!(\Sigma_{\setminus a})) \in \mathcal{M}_\Sigma$ is an example of a possible model that restricts a to occur only once, and thus, $\exists_{M\in\mathcal{M}_\Sigma}(\Sigma^*_{\setminus a}\circledast\boldsymbol{a}\{2,\})\cap\phi(M)=\emptyset$.
- Traces with exactly one occurrence of a: The set $(\Sigma^*_{\setminus a} \circledast \boldsymbol{a})$ contains all traces where a occurs exactly once. Since all process tree models are sound, there must exist at least one trace containing a. Additionally, because the absence or multiple occurrences of a are optional, all process models contain at least one trace with exactly one occurrence of a. Thus, $\forall_{M\in\mathcal{M}_\Sigma}(\Sigma^*_{\setminus a}\circledast\boldsymbol{a})\cap\phi(M)\neq\emptyset$.

The third group of traces, specifies that considering any activity $a \in \Sigma$, all the discovered models allow for at least one trace from $(\Sigma^*_{\setminus a} \circledast \boldsymbol{a})$.

Lemma 2. *Let $\Sigma \subseteq \mathcal{A}$ be a set of activities, Σ^* is all the traces we can generate over this set and $\mathcal{M}_\Sigma$ is the set of all process tree models we can generate with these activities.*

$$\forall_{M\in\mathcal{M}_\Sigma}\ \forall_{a\in\Sigma}\ \exists_{\sigma\in(\Sigma^*_{\setminus a}\circledast\boldsymbol{a})}\ \sigma\in\phi(M)$$

Proof. We proceed by structural induction on the process-tree model M.

Base case. If M is a single leaf labeled with some activity $b \in \Sigma$, then $\phi(M) = \boldsymbol{b}$. For the statement we must show that for each $a \in \Sigma$, there is a trace in $\Sigma^*_{\setminus a} \circledast \boldsymbol{a}$ belonging to $\phi(M)$.
- If $b = a$, then $\phi(M) = \boldsymbol{a}$ and clearly $a \in \Sigma^*_{\setminus a} \circledast \boldsymbol{a}$, so the claim holds.
- If $b \neq a$, then $\phi(M) = \boldsymbol{b}$ does not itself contain any a.

However, $\mathcal{M}_\Sigma$ contains only trees whose leaves include every activity in Σ at least once based on Definition 2, under that convention, the base case applies only when $b = a$.

Inductive step. Suppose the lemma holds for all proper subtrees of M. We show it for M by case distinction on the root operator $\oplus$ of $M = \oplus(M_1, M_2)$:

1. **Sequence** $\oplus = \rightarrow$: $\phi(M) = \rightarrow_\phi \big(\phi(M_1), \phi(M_2)\big) = \phi(M_1) \cdot \phi(M_2)$.

Let $a \in \Sigma$. By construction of process trees, a appears in exactly one subtree, say M_i where $i \in \{1,2\}$. By the induction hypothesis, there is a trace $\sigma \in \phi(M_i) \cap \big(\Sigma^*_{\setminus a} \circledast \boldsymbol{a}\big)$. For $j \neq i$, pick any $\sigma' \in \phi(M_j)$, since $a \notin act(M_j)$, we have $\sigma' \in \Sigma^*_{\setminus a}$. Then, $\sigma'' = \sigma.\sigma'$ if $i = 1$ and $\sigma'' = \sigma'.\sigma$ if $i = 2$ lies in $\phi(M)$ and contains exactly one a, so $\sigma \in \Sigma^*_{\setminus a} \circledast \boldsymbol{a}$.

2. **Exclusive choice** $\oplus = \times$: $\phi(M) = \phi(M_1) \cup \phi(M_2)$.

Since a labels a leaf in exactly one subtree M_i where $i \in \{1,2\}$, by the induction hypothesis there is $\sigma \in \phi(M_i) \cap (\Sigma^*_{\setminus a} \circledast \boldsymbol{a})$. Hence $\sigma \in \phi(M)$, as required.

3. **Concurrency** $\oplus = \wedge$: $\phi(M) = \wedge_\phi\big(\phi(M_1), \phi(M_2)\big) = \phi(M_1) \circledast \phi(M_2)$.

Again, let M_i be the unique subtree containing a where $i \in \{1,2\}$. By induction hypothesis pick

$$\sigma \in \phi(M_i) \cap (\Sigma^*_{\setminus a} \circledast \boldsymbol{a}), \quad \sigma' \in \phi(M_j) \subseteq \Sigma^*_{\setminus a} \quad (j \neq i).$$

Then any shuffle interleaving of σ and σ' produces a trace in $\phi(M)$ containing exactly one a, so the property holds.

4. **Loop** $\oplus = \circlearrowleft$: $\phi(M) = \circlearrowleft_\phi \big(\phi(M_1), \phi(M_2)\big) = \phi(M_1)\big(\phi(M_2) \cdot \phi(M_1)\big)^*$.

By tree well-formedness, a appears in exactly one of the subtrees. If $a \in M_1$, then by induction hypothesis there is $\sigma \in \phi(M_1) \cap (\Sigma^*_{\setminus a} \circledast \boldsymbol{a})$, and since $(\cdots)^*$ admits the empty iteration, $\sigma_1 \in \phi(M)$. If instead $a \in M_2$, then by induction hypothesis pick

$$\sigma' \in \phi(M_2) \cap (\Sigma^*_{\setminus a} \circledast \boldsymbol{a}), \quad \sigma \in \phi(M_1) \subseteq \Sigma^*_{\setminus a},$$

Thus $\sigma.\sigma'.\sigma \in \phi(M)$ and $\sigma.\sigma'.\sigma \in \Sigma^*_{\setminus a} \circledast \boldsymbol{a}$.

In every case, we have produced a trace in $\phi(M)$ with exactly one occurrence of a, which completes the induction.

Based on this property we define the common modes.

Definition 17 (*Common Modes*)**.** Let $\Sigma \subseteq \mathcal{A}$ be a subset of activities. Given that Σ^* is the set of all possible traces we can generate with activities in Σ, we define the common modes of Σ^* as follows,

$$com(\Sigma^*) = \{(\Sigma^*_{\setminus a} \circledast \boldsymbol{a}) | a \in \Sigma\}$$

According to Lemma 2, for any process model $M \in \mathcal{M}_\Sigma$ that can be generated with activities in Σ, and any $(\Sigma^*_{\setminus a} \circledast \boldsymbol{a}) \in com(\Sigma^*)$ allow for at least one trace that is allowed by this model. We write the equation of this lemma based on the definition of common modes as

$$\forall_{M\in\mathcal{M}_\Sigma}\forall_{\Lambda\in com(\Sigma^*)}\exists_{\sigma\in\Lambda}\sigma\in\phi(M)$$

Considering R as the set of rules, based on Lemma 2, if there is a common mode for which all traces violate the rule $r \in R$, we can conclude all the models have a trace from this common mode violate this rule. Therefore, all of the models $M \in \mathcal{M}_\Sigma$ violating the rule r (cf. Definition 9).

Lemma 3. *Let $\Sigma \subseteq \mathcal{A}$ be a set of activities, and $R \subseteq \mathcal{R}_{decl}$ be a set of rules. If there is a common mode $\Lambda \in com(\Sigma^*)$ such that all traces $\sigma \in \Lambda$ violate the rule $r \in R$, i.e., $\sigma \not\models r$, we can conclude all the models $M \in \mathcal{M}_\Sigma$ have a trace from Λ based on Lemma 2, and therefore all the models violate this rule, i.e., $M \not\models r$.*

$$\exists_{\Lambda \in com(\Sigma^*)} \forall_{\sigma \in \Lambda} \sigma \not\models r \Rightarrow \forall_{M \in \mathcal{M}_\Sigma} M \not\models r$$

Proof. Assume there is a common mode $\Lambda \in com(\Sigma^*)$ such that $\forall \sigma \in \Lambda : \sigma \not\models r$. We show that an arbitrary model $M \in \mathcal{M}_\Sigma$ must also violate r.

By Lemma 2, for every $M \in \mathcal{M}_\Sigma$ and every common mode Λ, there exists at least one trace $\sigma' \in \Lambda \cap \phi(M)$. Since by assumption $\sigma' \not\models r$, it follows that $\phi(M)$ contains a trace that does not satisfy r. By Definition 9, $M \not\models r$ whenever $\exists \sigma \in \phi(M) : \sigma \not\models r$. Hence M violates the rule r. Because M is chosen arbitrarily, this holds for all $M \in \mathcal{M}_\Sigma$, completing the proof.

We need to extend the common mode concept to the binary cuts in order to be able to prune the candidate cuts. $\phi_\cup$ is the maximum allowed behavior by models that can be discovered with cut c. Consider the union of all languages for a specific cut $c = (\oplus, \Sigma_1, \Sigma_2)$. According to Lemma 1:

$$\phi_\cup((\oplus, \Sigma_1, \Sigma_2)) = \oplus_\phi(\Sigma_1^*, \Sigma_2^*)$$

Building on the process tree semantics for various operators introduced in Definition 3, we define *common modes* as subsets of $\phi_\cup(c)$ such that, for all models $M \in \mathcal{M}_c$, at least one trace from the common mode exists in $\phi(M)$. Given $\Lambda_1 \in com(\Sigma_1^*)$ and $\Lambda_2 \in com(\Sigma_2^*)$, the operator $\oplus_\phi(\Lambda_1, \Lambda_2)$ generates a common mode for the cut c (cf. Definition 3 for details on how languages are generated for different operators). For the loop operator $\circlearrowleft$, the "do" part of the loop must occur at least once. However, each time the "do" part is executed, the "redo" part becomes active and may also occur. This repetition of the "do" and "redo" parts allows for the occurrence of different common modes in each iteration. Therefore, for the loop operator, we extend the definition of common modes to encompass a broader range of possibilities, i.e., two common modes $\Lambda_1, \Lambda_2 \in com(\Sigma_1^*)$ may follow each other, similarly, $\Lambda_3, \Lambda_4 \in com(\Sigma_2^*)$ may also follow each other.

Definition 18 (*Common Modes of a Cut*)**.** Let $L \in \mathcal{L}$ be an event log. $\mathcal{G}(L) = (\Sigma, E)$ is the corresponding DFG. Considering $(\oplus, \Sigma_1, \Sigma_2) \in C_\Sigma$ as a binary cut, we define the common modes of this cut as

- If $\oplus \in \{\rightarrow, \times, \wedge\}$

$$com\text{-}cut((\oplus, \Sigma_1, \Sigma_2)) = \{\oplus_\phi(\Lambda_1, \Lambda_2) | \Lambda_1 \in com(\Sigma_1^*), \Lambda_2 \in com(\Sigma_2^*)\}$$

- If $\oplus = \circlearrowleft$, Since the loop cut allow for infinite repetition of the redo part, each time a different common mode of do part or redo part can occur. The set of possibilities is very large. In order to check the satisfiability of the common modes with regards to the declarative rules, we need a finite subset of them.

$$\begin{aligned} com\text{-}cut((\oplus, \Sigma_1, \Sigma_2)) = &\{\Lambda | \Lambda \in com(\Sigma_1^*)\} \cup \\ &\{\Lambda_1.\Sigma_2^*.\Lambda_2 | \Lambda_1, \Lambda_2 \in com(\Sigma_1^*)\} \cup \\ &\{\Lambda_1.\Lambda_2.\Sigma_1^* | \Lambda_1 \in com(\Sigma_1^*), \Lambda_2 \in com(\Sigma_2^*)\} \cup \\ &\{\Sigma_1^*.\Lambda_1.\Lambda_2 | \Lambda_2 \in com(\Sigma_1^*), \Lambda_1 \in com(\Sigma_2^*)\} \cup \\ &\{\Sigma_1^*.\Lambda_1.\Sigma_1^*.\Lambda_2.\Sigma_1^* | \Lambda_1, \Lambda_2 \in com(\Sigma_2^*)\} \end{aligned}$$

If there is a common mode of a cut $\Lambda \in com\text{-}cut(c)$ for which all traces $\sigma \in \Lambda$ violate the rule $r \in R$, we can conclude all the models $M \in \mathcal{M}_c$ allow for a trace from this common mode Λ and therefore the cut violates the rule r.

Lemma 4. *Let $L \in \mathcal{L}$ be an event log. $\mathcal{G}(L) = (\Sigma, E)$ is the corresponding DFG. Let $R \subseteq \mathcal{R}_{decl}$ be a set of rules. Considering $(\oplus, \Sigma_1, \Sigma_2) \in C_\Sigma$, if there is a common mode $\Lambda \in com\text{-}cut(c)$ such that all traces $\sigma \in \Lambda$ violate the rule $r \in R$, i.e., $\sigma \not\models r$, we can conclude all the models $M \in \mathcal{M}_c$ have a trace from Λ based on Lemma 2, and therefore all the models violate this rule, i.e., $M \not\models r$.*

$$\exists_{\Lambda \in com\text{-}cut(c)} \forall_{\sigma \in \Lambda} \sigma \not\models r \Rightarrow \forall_{M \in \mathcal{M}_c} M \not\models r \Rightarrow c \not\models r$$

Proof. The proof proceeds in two steps.

Step 1: Every model $M \in \mathcal{M}_c$ admits a trace from any $\Lambda \in com\text{-}cut(c)$. By construction of $com\text{-}cut(c)$, each Λ is obtained as $\Lambda = \oplus_\phi(\Lambda_1, \Lambda_2)$ or, in the loop case, by one of the specified concatenations of $\Lambda_1, \Lambda_2, \Sigma_1^*, \Sigma_2^*$, in Definition 18 with $\Lambda_1 \in com(\Sigma_1^*)$ and $\Lambda_2 \in com(\Sigma_2^*)$.

Meanwhile,

$$\mathcal{M}_c = \left\{ \oplus(M_1, M_2) \;\middle|\; M_1 \in \mathcal{M}_{\Sigma_1},\ M_2 \in \mathcal{M}_{\Sigma_2} \right\}.$$

By Lemma 2, for each model $M_i \in \mathcal{M}_{\Sigma_i}$ where $i \in \{1, 2\}$, there is a trace $\sigma_i \in \phi(M_i) \cap \Lambda_i$. Moreover, whenever Λ is built, the semantics of $\oplus_\phi$ guarantees that one can assemble from σ_1 and σ_2 a trace $\sigma \in \phi\big(\oplus(M_1, M_2)\big) = \phi(M)$ such that $\sigma \in \Lambda$. Hence for every $M \in \mathcal{M}_c$ and every $\Lambda \in com\text{-}cut(c)$, $\phi(M) \cap \Lambda \neq \emptyset$.

Step 2: Violation of the rule. Assume there is some $\Lambda \in com\text{-}cut(c)$ such that $\forall \sigma \in \Lambda : \sigma \not\models r$. Take an arbitrary model $M \in \mathcal{M}_c$. By Step 1, there is $\sigma' \in \phi(M) \cap \Lambda$. But $\sigma' \not\models r$, therefore, by Definition 9 $M \not\models r$.

Since M was chosen arbitrarily, it follows that $\forall M \in \mathcal{M}_c : M \not\models r$. Therefore, the cut c itself cannot satisfy r, i.e., $c \not\models r$.

For each cut type, our goal is to identify common modes. Below, we provide a case-by-case distinction and detail the common modes for each cut type:

- **Sequence**: Considering $a \in \Sigma_1$ and $b \in \Sigma_2$, in all such models, there is at least one trace that has a first and then b. $(\Sigma^*_{1\setminus a} \circledast \boldsymbol{a}).(\Sigma^*_{2\setminus b} \circledast \boldsymbol{b})$ is the set of all such traces.

$$\textit{com-cut}((\rightarrow, \Sigma_1, \Sigma_2)) = \{(\Sigma^*_{1\setminus a} \circledast \boldsymbol{a}).(\Sigma^*_{2\setminus b} \circledast \boldsymbol{b}) | a \in \Sigma_1 \wedge b \in \Sigma_2\}$$

- **Exclusive choice**: Considering $a \in \Sigma_1$, $(\Sigma^*_{1\setminus a} \circledast \boldsymbol{a})$ is the set of all traces with activity a occurring once, and considering $b \in \Sigma_2$, $(\Sigma^*_{1\setminus b} \circledast \boldsymbol{b})$ contains all traces with activity b that can be generated. In all such models, only activities in Σ_1 can happen or activities in Σ_2.

$$\begin{aligned}\textit{com-cut}((\times, \Sigma_1, \Sigma_2)) = &\{(\Sigma^*_{1\setminus a} \circledast \boldsymbol{a}) | a \in \Sigma_1\} \cup \\ &\{(\Sigma^*_{2\setminus b} \circledast \boldsymbol{b}) | b \in \Sigma_2\}\end{aligned}$$

- **Concurrency**: Considering $a \in \Sigma_1$ and $b \in \Sigma_2$, $(\Sigma^*_{1\setminus a} \circledast \boldsymbol{a}) \circledast (\Sigma^*_{2\setminus b} \circledast \boldsymbol{b})$ allow for activity a occurring before b or activity b occurring before a.

$$\textit{com-cut}((\wedge, \Sigma_1, \Sigma_2)) = \{(\Sigma^*_{1\setminus a} \circledast \boldsymbol{a}) \circledast (\Sigma^*_{2\setminus b} \circledast \boldsymbol{b}) | a \in \Sigma_1 \wedge b \in \Sigma_2\}$$

- **Loop**: Considering $a \in \Sigma$, all the models should allow for a subset of traces from $(\Sigma^*_{1\setminus a} \circledast \boldsymbol{a})$ meaning that activity a should be able to occur only once. Considering $a_1, a_2 \in \Sigma_1$ all the models allow for a subset of traces in $(\Sigma^*_{1\setminus a_1} \circledast \boldsymbol{a_1}).\Sigma^*_2.(\Sigma^*_{1\setminus a_2} \circledast \boldsymbol{a_2})$ indicating any two activities in Σ_1 can eventually follow each other. Considering $b_1, b_2 \in \Sigma_2$, any activity in Σ_1 can be followed by any activity in Σ_2 represented with $(\Sigma^*_{1\setminus a_1} \circledast \boldsymbol{a_1}).(\Sigma^*_{2\setminus b_1} \circledast \boldsymbol{b_1}).\Sigma^*_1$ and $\Sigma^*_1.(\Sigma^*_{2\setminus b_1} \circledast \boldsymbol{b_1}).(\Sigma^*_{1\setminus a_1} \circledast \boldsymbol{a_1})$. In addition, the re-do part of the loop can occur several times which indicates all the models allow for activities in Σ_2 to eventually be followed by each other denoted with $\Sigma^*_1.(\Sigma^*_{2\setminus b_1} \circledast \boldsymbol{b_1}).\Sigma^*_1.(\Sigma^*_{2\setminus b_2} \circledast \boldsymbol{b_2}).\Sigma^*_1$. In summary, $\textit{com-cut}(\circlearrowleft, \Sigma_1, \Sigma_2)$ included the below listed modes.

$$\begin{aligned}\textit{com-cut}((\circlearrowleft, \Sigma_1, \Sigma_2)) = &\{(\Sigma^*_{1\setminus a} \circledast \boldsymbol{a}) | a \in \Sigma_1\} \cup \\ &\{(\Sigma^*_{1\setminus a_1} \circledast \boldsymbol{a_1}).\Sigma^*_2.(\Sigma^*_{1\setminus a_2} \circledast \boldsymbol{a_2}) | a_1, a_2 \in \Sigma_1\} \cup \\ &\{(\Sigma^*_{1\setminus a_1} \circledast \boldsymbol{a_1}).(\Sigma^*_{2\setminus b_1} \circledast \boldsymbol{b_1}).\Sigma^*_1 | a_1 \in \Sigma_1 \wedge b_1 \in \Sigma_2\} \cup \\ &\{\Sigma^*_1.(\Sigma^*_{2\setminus b_1} \circledast \boldsymbol{b_1}).(\Sigma^*_{1\setminus a_1} \circledast \boldsymbol{a_1}) | a_1 \in \Sigma_1 \wedge b_1 \in \Sigma_2\} \cup \\ &\{\Sigma^*_1.(\Sigma^*_{2\setminus b_1} \circledast \boldsymbol{b_1}).\Sigma^*_1.(\Sigma^*_{2\setminus b_2} \circledast \boldsymbol{b_2}).\Sigma^*_1 | b_1, b_2 \in \Sigma_2\}\end{aligned}$$

Examples of conflicting common modes

Considering two activities $a, b \in \Sigma$, in Table 1 the rules that violate are reported. The rows are the declarative constraints including these two activities (if the template has single parameter, only a), the columns specify considering cut $c = (\oplus, \Sigma_1, \Sigma_2)$ to with subsets Σ_1 or Σ_2 belong these two activities. The cell of the table indicates the subset of cut type operators that $c = (\oplus, \Sigma_1, \Sigma_2)$ is violated by the corresponding declarative rule.

As an example for some rules and cut types we show an example of a common mode conflicting.

- $AtLeast1(a)$ and $(\circlearrowleft, \{c, d\}, \{a, b\})$: $(\boldsymbol{d})^*\boldsymbol{c}(\boldsymbol{d})^*$ in all such traces, activity a does not exist.
- $Response(a, b)$ and $(\times, \{a, c\}, \{b, d\})$: $(\boldsymbol{c})^*\boldsymbol{a}(\boldsymbol{c})^*$ in all such traces, activity a occurs, but b does not occur after it.
- $NotSuccession(a, b)$ and $(\wedge, \{b, c\}, \{a, d\})$: $(\boldsymbol{c}|\boldsymbol{d})^*\boldsymbol{a}(\boldsymbol{c}|\boldsymbol{d})^*\boldsymbol{b}(\boldsymbol{c}|\boldsymbol{d})^*$ in all such traces, activity a occurs, and b occurs after it.

The declarative template $Absence(a)$ specifies that activity a must not occur in any trace allowed by the model. Since the discovered models are sound there is at least one trace that contains activity a. Therefore, it is impossible to discover a model without violation of this rule unless, activity a is not included in the model. Therefore, in case the declarative rule $Absence(a)$ is in R, we filter out activity a from the event log and discover the process model for the filtered event log.

7. Evaluation

The proposed framework has been fully implemented and is publicly available[2]. In this research, we had the opportunity to collaborate with UWV, a Dutch insurance agency. The event log provided from one of their claim-handling processes, along with the input from domain experts, enabled us to evaluate our framework using domain-driven rules. In addition to this domain-based evaluation, we conducted further experiments on publicly available event logs. For these logs, since domain knowledge is not directly available, we derive declarative constraints using the Minerful algorithm [17]. However, the IMr framework is designed to operate with any input rule $r \in \mathcal{R}_{decl}$. The declarative rules discovered by a specific technique are utilized to assess the ability of our discovery framework to guide the process of discovering models that satisfy these rules.

In this section, we aim to achieve the following goals:

[2] https://github.com/aliNorouzifar/rule_guided_process_discovery.

Table 1
Violation of declarative constraints for two activities $a, b \in \Sigma$ across different cut configurations $c = (\oplus, \Sigma_1, \Sigma_2)$.

	$a \in \Sigma_1, b \in \Sigma_1$	$a \in \Sigma_2, b \in \Sigma_2$	$a \in \Sigma_1, b \in \Sigma_2$	$a \in \Sigma_2, b \in \Sigma_1$
$AtLeast1(a)$	{×}	{↺, ×}	{×}	{↺, ×}
$AtMost1(a)$	{↺}	{↺}	{↺}	{↺}
$Absence(a)$	{∧, ↺, ×, →}	{∧, ↺, ×, →}	{∧, ↺, ×, →}	{∧, ↺, ×, →}
$Init(a)$	{∧, ×}	{∧, ↺, ×, →}	{∧, ×}	{∧, ↺, ×, →}
$End(a)$	{∧, ×, →}	{∧, ×, ↺}	{∧, ×, →}	{∧, ×, ↺}
$RespondedExistence(a, b)$			{↺, ×}	{×}
$CoExistence(a, b)$			{↺, ×}	{↺, ×}
$Response(a, b)$			{∧, ×, ↺}	{∧, ×, →}
$AlternateResponse(a, b)$			{∧, ×, ↺}	{∧, ×, →}
$ChainResponse(a, b)$			{∧, ×, ↺}	{∧, ×, →}
$Precedence(a, b)$			{∧, ×}	{∧, ↺, ×, →}
$AlternatePrecedence(a, b)$			{∧, ×}	{∧, ↺, ×, →}
$ChainPrecedence(a, b)$			{∧, ×}	{∧, ↺, ×, →}
$Succession(a, b)$			{∧, ×, ↺}	{∧, ↺, ×, →}
$AlternateSuccession(a, b)$			{∧, ×, ↺}	{∧, ↺, ×, →}
$ChainSuccession(a, b)$			{∧, ×, ↺}	{∧, ↺, ×, →}
$NotRespondedExistence(a, b)$	{↺}	{↺}	{∧, ↺, →}	{∧, ↺, →}
$NotCoExistence(a, b)$	{↺}	{↺}	{∧, ↺, →}	{∧, ↺, →}
$NotResponse(a, b)$	{↺}	{↺}	{∧, ↺, →}	{∧, ↺}
$NotChainResponse(a, b)$				
$NotPrecedence(a, b)$	{↺}	{↺}	{∧, ↺, →}	{∧, ↺}
$NotChainPrecedence(a, b)$				
$NotSuccession(a, b)$	{↺}	{↺}	{∧, ↺, →}	{∧, ↺}
$NotChainSuccession(a, b)$				

- **Case study with domain knowledge:** Leveraging domain knowledge and real-world event data from a claim-handling process of the UWV agency, we demonstrate how rules derived from domain expertise help discover process models that align more closely with expected behavior.
- **Rule satisfaction:** Regardless of the parameter settings that generate the rules, our algorithm demonstrates the capability to satisfy more rules compared to baseline models that do not utilize any rules as input.
- **Model quality:** It is crucial to maintain the quality of the discovered models while incorporating the rules. Models that satisfy the rules but exhibit low fitness and precision are undesirable. Therefore, we compare our results with IMbi [8], IMf [20], and Split Miner [23] to ensure both rule satisfaction and high model quality.
- **Computational cost:** The computational cost of IMbi, which is a similar approach without rule incorporation, is high due to the large candidate cut set. We show that introducing rules to the algorithm, as discussed in this paper, can reduce computational costs. However, IMf and Split Miner still perform better in terms of efficiency, as they avoid exploring many options in intermediate steps.

7.1. Event logs

Several real-life event logs, widely recognized in the process mining community, are utilized to evaluate our framework. Alongside publicly available event logs, we conduct experiments with a claim-handling process from the UWV employee insurance agency in the Netherlands. For this case, we had the opportunity to gather domain expert feedback, allowing us to tailor the framework to produce realistic and meaningful results. These event logs present diverse and challenging scenarios, effectively testing the robustness and performance of the framework. The characteristics of the event logs, including the number of activities, events, traces, and trace variants, are summarized in Table 2. The event logs used in this study are detailed as follows:

- **BPI Challenge 2012 (BPIC12)** [30]**:** This log originates from a Dutch financial institution and focuses on a loan application process. For our evaluation, we only include the *application* and *offer* subprocesses, which represent key stages in the process.
- **BPI Challenge 2017 (BPIC17)** [31]**:** This dataset pertains to a loan application process at another financial institution. Similar to BPIC12, we include only the *application* and *offer* subprocesses to reduce complexity while retaining relevant behavior.
- **BPI Challenge 2018 (BPIC18)** [32]**:** This event log covers the handling of applications for EU direct payments to German farmers from the European Agricultural Guarantee Fund. The process repeats annually with minor changes due to evolving EU regulations. For our evaluation, we focus on the *application* subprocess.
- **Road Traffic Fine Management (RTFM)** [33]**:** This log contains events related to the processing of road traffic fines by an Italian governmental organization. It provides a structured yet realistic process scenario involving fine issuance, notifications, and payments.
- **Sepsis** [34]**:** The Sepsis log originates from a hospital and captures the treatment processes of patients with sepsis. It includes events such as admission, diagnosis, laboratory testing, and treatment, making it an excellent dataset for healthcare process analysis.

Table 2
Event logs used in experiments.

	#activities	#events	#traces	#trace variants
BPIC 12	17	92,093	13,087	576
BPIC 17	18	433,444	31,509	2630
BPIC 18	15	928,091	43,809	1435
RTFM	11	561,470	150,370	231
Hospital	18	451,359	100,000	1020
Sepsis	16	15,214	1050	846
UWV	16	1,309,719	144,046	484
UWV_filtered	16	194,313	20,112	483

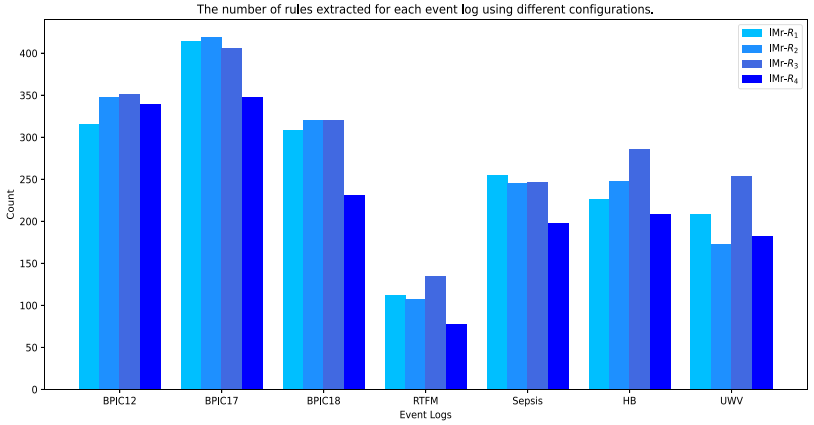


Fig. 6. Total number of declarative rules discovered for each configuration (R_1, R_2, R_3, and R_4) using the Minerful algorithm across different event logs.

- **Hospital Billing (HB)** [35]**:** This dataset focuses on the billing procedures within a hospital environment. It includes various activities related to the administrative handling of patient bills, such as bill creation, validation, and payment.
- **UWV:** This log comes from the Dutch Employee Insurance Agency (UWV) and captures processes related to handling benefit claims. It is particularly useful for analyzing administrative and service-oriented workflows.
- **UWV_filtered:** The original event log, UWV, contains a dominant trace variant with a frequency of 86%. This predominance can heavily influence the evaluation results, potentially biasing the analysis. To address this, we introduce a filtered version of the event log, UWV_filtered, where the dominant trace variant is excluded. This filtered event log is incorporated into experiments focused on assessing the quality of the discovered process models, enabling a more robust evaluation.

7.2. Experimental setup

Declarative rules

We utilize the *Minerful* discovery algorithm, a widely recognized method for declarative process discovery, to extract declarative rules using various parameter settings [17]. Specifically, the *Minerful* algorithm is executed under four distinct configurations, each defined by varying values of the *support* and *confidence* parameters. These configurations are designed to generate diverse rule sets for evaluating our process discovery technique:

- R_1: $confidence = 0.95$, $support = 0.030$
- R_2: $confidence = 0.98$, $support = 0.010$
- R_3: $confidence = 0.99$, $support = 0.005$
- R_4: $confidence = 1.00$, $support = 0.000$

The total number of rules discovered for each configuration and event log is illustrated in Fig. 6.

To assess the similarity between the rule sets obtained from these configurations, we summarize their pairwise similarities in Fig. 7. The similarity between two rule sets is quantified using the following metric, which measures the proportion of rules in the intersection relative to the union of the rule sets:

$$\text{similarity}(R_i, R_j) = \frac{|R_i \cap R_j|}{|R_i \cup R_j|}.$$

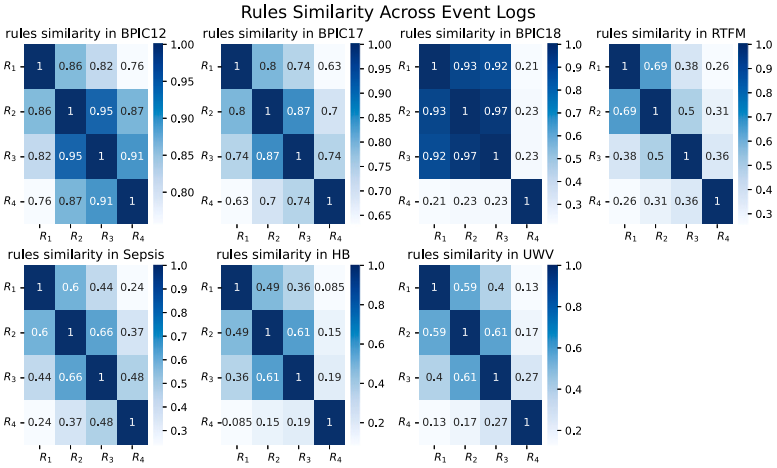


Fig. 7. Pairwise similarity of rule sets discovered under different configurations (R_1, R_2, R_3, and R_4), measured as the proportion of shared rules relative to the total number of rules.

This metric facilitates a comprehensive comparison by considering both shared and unique rules across different configurations. The objective is to emphasize the diversity of the generated rule sets, showcasing that the evaluation of our discovery technique is conducted using a variety of inputs. Through experiments, we demonstrate that our algorithm maintains robust performance regardless of the specific parameter settings employed to generate the rules.

Process discovery algorithms

Numerous process discovery techniques have been proposed in the literature. However, comparative studies have shown that not all of them perform well when applied to complex or large-scale event logs [2]. These methods also vary significantly in their theoretical guarantees, such as soundness, and in the types of process models they produce. Among the well-established techniques, those that generate block-structured models have demonstrated better understandability and applicability in real-life settings. This is further evidenced by the adoption of the inductive miner and split miner discovery techniques in commercial tools such as Celonis and SAP Signavio, respectively, highlighting their practical relevance. In the extensive benchmark study presented in [2], the inductive miner, split miner, and evolutionary tree miner are identified as top-performing techniques based on a comprehensive set of evaluation criteria. However, we excluded Evolutionary Tree Miner from our baseline comparisons due to its limited scalability, stemming from its genetic algorithm-based nature. Consequently, we selected the inductive miner and the split miner as representative and competitive baselines for our evaluation, given their strong performance, theoretical foundations, and demonstrated applicability in real-world scenarios. The discovery algorithms used in this study and their corresponding parameters are as follows:

- **IMr:** This algorithm has a single parameter, *support* $\in [0, 1]$. We vary this parameter in intervals of 0.1, ranging from 0.1 to 0.9.
- **IMbi** [8]**:** IMbi includes two parameters *support* $\in [0, 1]$ and *ratio* $\in [0, 1]$. The *ratio* parameter controls the impact of undesirable event logs during process discovery. Since we only have one event log in this study, we set *ratio* $= 0$ to effectively ignore undesirable logs. IMbi, when used without the undesirable event log, is functionally equivalent to IMr with an empty rule set. Similar to IMr, the *support* parameter is varied in 0.1 intervals from 0.1 to 0.9.
- **IMf** [20]**:** This algorithm introduces an infrequency filtering parameter, $f \in [0, 1]$. We adjust this parameter in intervals of 0.1, ranging from 0.1 to 0.9.
- **Split Miner** [23]**:** Split Miner operates with two parameters $\epsilon \in [0, 1]$ and $\eta \in [0, 1]$. Based on the hyperparameter optimization performed in [23], the best-performing parameter settings are typically $\epsilon = 0.1$ and $\eta = 0.4$. For this study, we explore process models using similar settings, with ϵ set to either 0.1 or 0.2, and η varied in 0.1 intervals between 0.1 and 0.5.

Beyond evaluating rule satisfaction, it is crucial to preserve the conformance dimension of the discovered models, which can be assessed using fitness, precision, and the F1-score (the harmonic mean of fitness and precision).

Fitness measures the ability of a process model to replay the behavior observed in the event log. A trace $\sigma \in L$ is not replayable by the model if $\sigma \notin \phi(M)$. While this notion of fitness is binary, traces that are not replayable may still be similar to some traces in $\phi(M)$. To address this limitation, the concept of *optimal alignment* has been introduced in the literature. This approach measures how well a trace can be replayed by computing the alignment cost between the trace and the model. The resulting *alignment fitness* metric provides a fitness degree ranging from 0 to 1 [3]. Precision evaluates the ability of a process model to represent only the behavior observed in the event log. In this paper, we use the precision based on the escaping edges concept introduced in [3]. Since

both fitness and precision are critical for evaluating conformance, the F1-score is often used to capture the trade-off between these two measurements. The F1-score is calculated as the harmonic mean of alignment fitness and precision:

$$F1\text{-}score = \frac{2 \times fitness \times precision}{fitness + precision}.$$

Among the discovered models generated using various parameter configurations, we select the best models by identifying those with the maximum $F1\text{-}score$ while ensuring a $fitness$ value above 0.9. This criterion guarantees that the discovered models accurately represent the behavior observed in the event log while achieving a balanced trade-off between fitness and precision.

7.3. Analysis of the results

UWV case study

This study is conducted in collaboration with the UWV agency and involves the analysis of an event log from one of their claim-handling processes, covering data for 144,046 clients. Fig. 11(a) depicts the normative model derived from an extensive analysis of the event log, enriched by domain knowledge provided by process experts actively participating in this case study. The process begins when a claim is received. Some cases are blocked (*Block Claim 1*), after which corrections are made, and the case is subsequently unblocked. In other instances, a different type of blocking (*Block Claim 2*) occurs shortly after the claim is initiated. These cases typically lead to claim rejection, which may be followed by an objection from the client. If the claim is accepted, the client becomes eligible to receive one to three payments.

However, some clients may file objections after receiving one or more payments. In such cases, no further payments should be made. The activity *Block Claim 3* is used to stop the process from executing additional payments. It can be triggered either after receiving an objection (*Receive Objection 1*) that permanently stops further payments or, in some cases, due to internal reasons, such as incorrect payment details, that temporarily block further payments. These issues may later be resolved through the *Unblock Claim 3* activity.

When an objection is received, the claim is typically blocked by the agency. In parallel, the client should submit a withdrawal request and repay the previously received benefits. Since *Withdraw Claim* and *Repayment* are initiated by the client, while *Block Claim 3* is performed by the agency, the use of an AND-gateway in the BPMN model captures the concurrency between these activities, allowing them to occur in different execution orders.

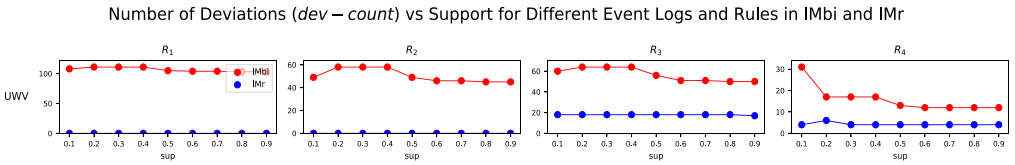
The knowledge we have about the process, along with the normative model developed in collaboration with domain experts, serves as the basis for deriving domain-informed declarative rules to guide the process discovery. Rather than providing an extensive, yet potentially incomplete, set of rules upfront, we follow a more iterative and targeted approach. First, we discover process models using standard techniques, then analyze and discuss these models with domain experts. The normative model acts as shared reference knowledge in this process, helping to identify shortcomings in the discovered models. From these discussions, we extract a more focused and meaningful set of rules aimed at addressing the specific deficiencies observed.

Based on the parameter settings described for the IMbi, IMf, and split miner process discovery techniques, the best-performing models, i.e., those with the highest $F1$-score, without using any input rules, are summarized as follows:

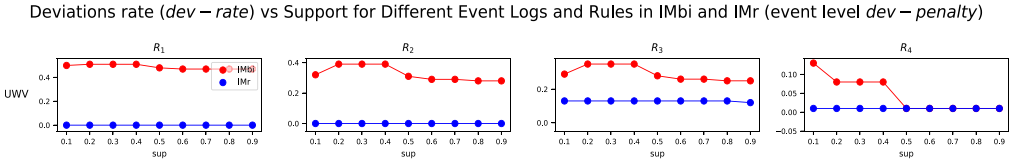
- IMbi: The best model illustrated in Fig. 11(d) is discovered with the parameter of $sup = 0.5$, this model has a $fitness$ of 0.99 and $precision$ of 0.90 and $F1\text{-}score$ of 0.95.
- IMf: The best model illustrated in Fig. 11(c) is discovered with parameter $f = 0.5$, this model have $fitness$ of 0.96, $precision$ of 0.94 and $F1\text{-}score$ of 0.95.
- Split Miner: The best model discovered by the split miner illustrated in Fig. 11(b) is achieved with $\epsilon = 0.1$ and $\eta = 0.1$. This model has a $fitness$ of 0.99 and $precision$ of 0.89 and $F1\text{-}score$ of 0.94.

After discovering process models using state-of-the-art process discovery techniques, we compared the normative model as the ground truth with those discovered using automated techniques, particularly the models generated by the IMbi method, as it shares the same core discovery algorithm. The discussions with domain experts, grounded in this comparative analysis, led to the formulation of the following declarative rules. These rules reflect important domain expectations that should be captured in a good process model but are only partially or insufficiently addressed in the discovered models.

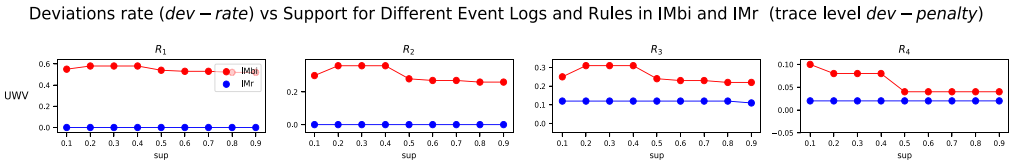
- *NotCoExistence(Block Claim 1, Reject Claim)*: The *Block Claim 1* mechanism is not intended to lead to the rejection of claims. Instead, it should be followed by corrections and eventual unblocking.
- *Precedence(Block Claim 3, Unblock Claim 3)*: Not every occurrence of *Block Claim 3* is followed by *Unblock Claim 3*, but unblocking can only happen if the claim was previously blocked.
- *Response(Unblock Claim 3, Payment Order)*: *Unblock Claim 3* indicates that the claim, previously blocked, is now cleared to proceed with the next payment order and its execution.
- *AtMost1(Withdraw Claim)*: Each claim in the process may only be withdrawn once.
- *AtMost1(Repayment)*: Each claim may only be subject to a single repayment.
- NotSuccession(Receive Objection 1, Payment Order): Although in the event log some payments are followed by the activity *Receive Objection 1* (in approximately 19% of the claims that include this activity), the agency asserts that this is not a desirable behavior. Such occurrences may result from technical limitations in the system's implementation. However, in a well-structured process model, we aim to ensure that *Receive Objection 1* is not followed by any further payments.



(a) Evaluation metric *dev-count*.



(b) Evaluation metric *dev-rate* considering *support* of rules in the event level as *dev-penatly*.



(c) Evaluation metric *dev-rate* considering *support* of rules in the trace level as *dev-penatly*.

Fig. 8. The evaluation metrics *dev-count* and *dev-rate* are calculated for the models discovered using the IMbi and IMr techniques across different *sup* values, considering various rule sets: R_1, R_2, R_3, and R_4.

– NotSuccession(Withdraw Claim, Payment Order): After the withdrawal of a claim, no further payments should be processed. Although the blocking mechanism may be triggered after the withdrawal in some cases, no payments should be executed once the claim has been withdrawn.
– AlternateSuccession(Withdraw Claim, Repayment): Any withdrawal of a claim should be followed by the repayment of the previously received benefits.

This set of 8 declarative rules, referred to as R_d and derived from domain knowledge, is used alongside the input event log to guide the IMr process discovery technique toward generating more accurate models. Applying the IMr discovery technique with a support threshold of $sup = 0.2$, we obtained a process model that achieved the highest $F1$-score of 0.95, resulting from a fitness value of 1 (0.9974 before rounding) and a precision value of 0.91. The discovered model is depicted in Fig. 11(f).

Before conducting the comparative analysis of the discovered models and evaluating the strengths and weaknesses of each, we first performed an additional analysis to assess whether automated rule discovery techniques, such as Minerful, can generate meaningful rules to guide the process discovery. To this end, we applied the Minerful declarative process discovery technique with the configuration settings detailed in Section 7.2, enabling the automatic extraction of declarative constraints from the event log without relying on domain knowledge. Fig. 6 shows the number of rules discovered under different parameter configurations, while Fig. 7 illustrates the pairwise similarity between the resulting rule sets.

Comparison between IMr vs. IMbi. The IMbi algorithm with parameter $ratio = 0$ is equivalent to the IMr algorithm without any rules as input. We discovered process models for different parameter settings of IMbi and IMr algorithm. For each event log, the rule sets R_1, R_2, R_3, and R_4 are used as input besides the event log to investigate how well the algorithm can employ them. We used *dev-count* and *dev-rate* as defined in Definition 14 to evaluate the discovered model. Fig. 8 presents the evaluation metrics used to assess the compliance of each rule set with the final models discovered using the IMbi and IMr techniques across different parameter settings. Specifically, Fig. 8(a) shows the number of violated rules, denoted as *dev-count*, for each rule set. These violations are measured against the process models discovered using different values of the *sup* parameter, which are indicated on the x-axis.

While the number of deviating rules provides valuable insights, not all rules carry equal importance. The *support* parameter of Minerful (cf. Definition 15) represents the satisfaction rate of a constraint. Declarative constraints with low support indicate that the constraint is rarely observed in the event log, making its violation less significant. Conversely, constraints with higher support are satisfied more frequently and thus carry greater importance. Figs. 8(b) and 8(c) present the *dev-rate* metric, which is computed using the *support* value of individual rules as the *dev-penalty*, evaluated at the event and trace levels, respectively. The goal of this analysis is to demonstrate that IMr effectively incorporates the input rules and that these rules have a significant impact on the resulting process structure.

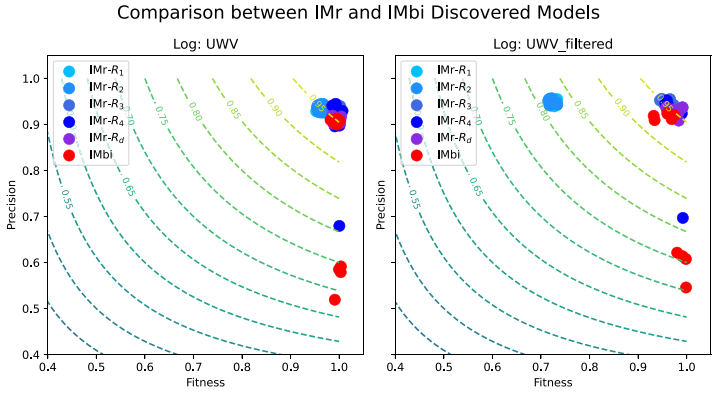


Fig. 9. Conformance checking metrics for models discovered from the UWV event log (left) and its filtered version excluding the most frequent trace variant (right). Each point represents a discovered model using a specific rule set (visualized with different colors) and different *sup* parameter values for the IMbi and IMr discovery techniques. (If you are viewing a black-and-white version of this paper, please refer to the online version to interpret the color references in this figure).

The next step is to evaluate the quality of the discovered models under different rule configurations using traditional metrics such as fitness, precision, and $F1\text{-}score$. However, the UWV event log contains a highly frequent trace variant, which may bias the results of conformance checking. To mitigate this effect, we used a filtered version of the event log, excluding the most frequent trace variant, to assess whether the discovered models can adequately capture and explain the remaining behavioral patterns.

Fig. 9 represents the conformance checking metrics for the UWV event log (left-hand side) and the filtered UWV event log which excludes the most frequent trace variant (right-hand side). In each subfigure, the x-axis represents fitness, the y-axis represents precision, and the dashed lines indicate $F1\text{-}score$, i.e., all points along the same dashed line share the same $F1\text{-}score$. The $F1\text{-}score$ increases as the points move toward the top-right corner of the plot, indicating better overall model quality. The red points in each figure correspond to the models discovered using the IMbi discovery algorithm. Different shades of blue represent the models discovered using the IMr technique under various rule configurations of the Minerful discovery technique, specifically, rule sets R_1, R_2, R_3, and R_4. The purple points indicate the models discovered using IMr with rules derived from domain knowledge, denoted as R_d.

A notable difference is observed in the conformance checking results between the original event log and the filtered event log, particularly under rule configurations R_1 and R_2. This discrepancy is primarily attributed to the presence of *Absence* rules with lower $confidence$ values. For instance, in R_1, the following *Absence* rules are included:

- *Absence*(Correct Claim), with $confidence = 0.953$
- *Absence*(Reject Claim), with $confidence = 0.962$
- *Absence*(Block Claim 1), with $confidence = 0.953$
- *Absence*(Unblock Claim 1), with $confidence = 0.953$
- *Absence* (Block Claim 2), with $confidence = 0.963$
- *Absence*(Block Claim 3), with $confidence = 0.956$
- *Absence*(Unblock Claim 3), with $confidence = 0.996$
- *Absence*(Withdraw Claim), with $confidence = 0.971$
- *Absence*(Repayment), with $confidence = 0.978$
- *Absence* (Receive Objection 1) with $confidence = 0.961$
- *Absence*(Receive Objection 2), with $confidence = 0.990$

The exclusion of these activities results in a simpler process model that achieves high conformance with respect to the most frequent trace variant of the UWV event log. However, this simplification comes at the cost of significantly poorer performance for other trace variants, excluding the most frequent one.

Comparison to other discovery algorithms. Fig. 10 compares the models discovered using IMbi, IMf, and split miner with those obtained via the IMr discovery technique, considering rule set R_3 (represented in blue) as the best-performing configuration among the automatically discovered rule sets, and R_d (represented in purple) as the domain-derived rule set. This quantitative evaluation shows that the models discovered using IMr achieve comparable or superior conformance checking values. This indicates that incorporating rules as input does not negatively impact model quality and can, in fact, lead to improved process models even beyond their better alignment with domain knowledge.

In Fig. 10, the conformance results for both the original event log and the filtered event log, where the most frequent trace variant has been excluded, are presented. While the IMf algorithm produces high-quality process models when using the original event log, the conformance checking results for the filtered event log highlight a limitation. Specifically, these models struggle to effectively

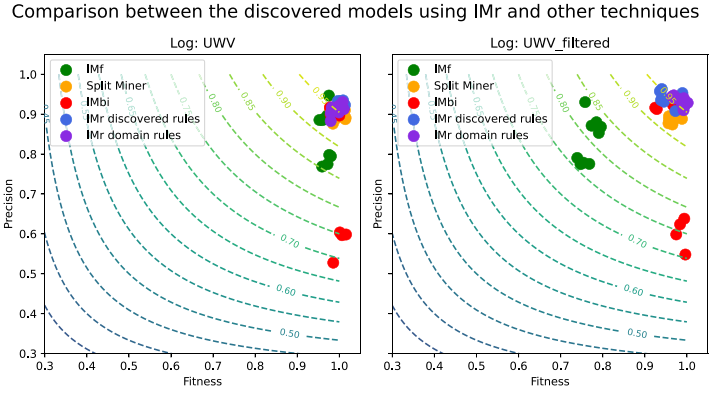
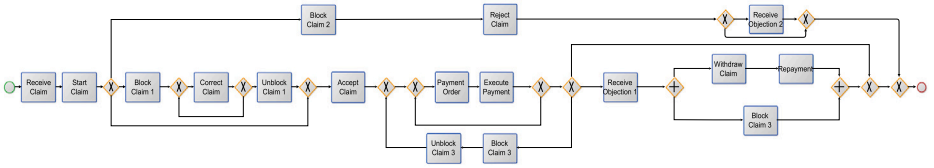
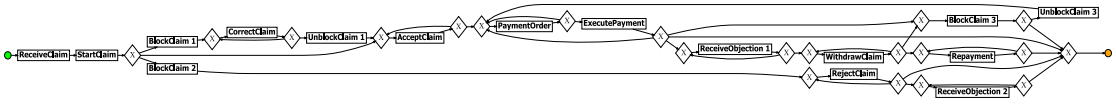


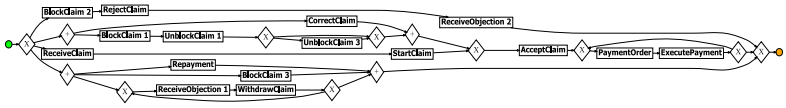
Fig. 10. Conformance checking metrics comparing models discovered using IMbi, IMf, and split miner with those obtained via the IMr technique. IMr is evaluated using rule set R_3 (blue), the best-performing among the automatically discovered sets, and R_d (purple), derived from domain knowledge. The color of each point indicates the discovery method used, while variations in parameter settings for each method result in multiple points of the same color. (If you are viewing a black-and-white version of this paper, please refer to the online version to interpret the color references in this figure).



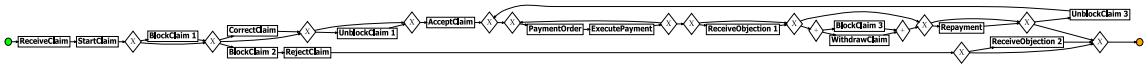
(a) The normative model of the UWV claim-handling process.



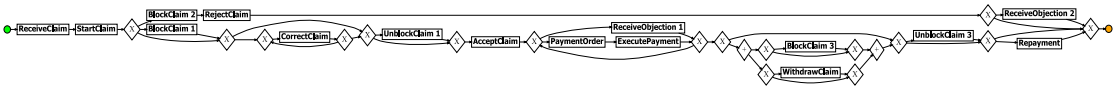
(b) Process model discovered using Split Miner with $\epsilon = 0.1$ and $\eta = 0.1$.



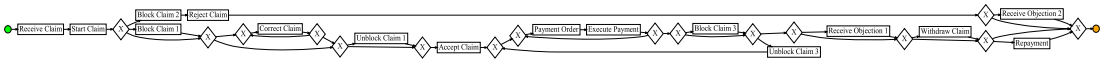
(c) Process model discovered using IMf with *sup*=0.5.



(d) Process model discovered using IMbi with *sup*=0.5.



(e) Process model discovered using IMr with *sup*=0.2 and discovered rules.



(f) Process model discovered using IMr with *sup*=0.2 and rules from domain knowledge.

Fig. 11. The best discovered models from UWV event log using different process discovery algorithms.

capture and represent trace variants beyond the most frequent one. In contrast, the other discovery techniques demonstrate the ability to represent both the original and the filtered event logs with greater robustness.

Analysis of discovered models. In addition to the quantitative analysis, we conducted a qualitative analysis of the discovered models based on the domain knowledge we had available through domain experts and the obtained normative model:

- Discovered model with split miner represented in Fig. 11(b): Several activities in the model are incorrectly involved in self-loops, which are not intended according to the process logic. Notable examples include *Accept Claim*, *Payment Order*, *Withdraw Claim*, and *Repayment*. Additionally, the structure representing the post-payment behavior lacks the clarity and straightforwardness typically found in models discovered using inductive miner techniques, making it more difficult for domain experts to interpret. In particular, the positioning of activities following the handling of payments for accepted cases differs significantly from the designed normative model shown in Fig. 11(a). The branching structure following *Block Claim 3*, defined by an exclusive choice gateway, only permits either unblocking the claim or terminating the process. However, the correct process behavior should also allow for the withdrawal of the claim and subsequent repayment of benefits as valid continuations. This process model incorrectly allows a claim to be withdrawn without requiring repayment of the received benefits, which should not be possible.
- Discovered model with IMf represented in Fig. 11(c): This process model performs the worst among the selected models, as it fails to position activities in their correct relative order. The model appears to have been heavily influenced by noise in the event log and seems to have been tailored to accommodate only the most frequent trace variants. We highlight a few of its major shortcomings: the only valid start activity should be *Receive Claim*, which must be followed by *Start Claim*, however, the model permits several other activities to initiate the process. Furthermore, the branching structure following the first exclusive choice disrupts the proper sequencing of activities. For example, activities such as *Block Claim 1* and *Unblock Claim 1* should appear after *Start Claim*, and *Block Claim 3*, *Repayment*, and *Withdraw Claim* should be after the execution of payments.
- Discovered model with IMbi represented in Fig. 11(d): The execution of *Block Claim 1* should be restricted to accepted claims. However, the current process model allows this activity to occur before the exclusive choice that separates the paths for accepted and rejected cases. Additionally, multiple occurrences of *Correct Claim* are not permitted in the model, although such repetitions are allowed in practice. There should be no payment order or execution after *Receive Objection 1*. However, this process model allows *Receive Objection 1* to be followed by additional payments. *Repayment* should only be possible if the claim has been withdrawn. The looping behavior introduced by *Unblock Claim 3* may occur without a blocking order and enables multiple executions of *Withdraw Claim* and *Repayment*, although each of these should occur at most once.
- Discovered model with IMr with discovered rules represented in Fig. 11(e): Although the conformance checking results show no significant difference between this model and the one discovered using domain-specific rules, this model exhibits less alignment with the normative model. Several deficiencies highlight this misalignment: *Receive Objection 1* may occur before the execution of any payments or after any payment, and it can be followed by further payments. This behavior is not desirable in a discovered model, as it contradicts the expected process logic. Additionally, the positioning of *Unblock Claim 3* is inaccurate. It should occur only after *Block Claim 3* and should precede any subsequent payments, reflecting the resolution of the issue that initially triggered the block. Moreover, this model incorrectly allows for the repayment of benefits without a corresponding claim withdrawal, as well as the withdrawal of a claim without requiring repayment.
- Discovered model with IMr with domain rules represented in Fig. 11(f): A visual comparison between this model and the normative model shown in Fig. 11(a) reveals a good level of alignment with the expected process behavior. Based on the evaluation by process experts, this model provides the most accurate representation of the process among the discovered models. However, certain deficiencies remain that IMr is unable to address effectively, primarily due to its limitations in capturing long-term dependencies. For instance, *Unblock Claim 1* should only occur if it is preceded by *Block Claim 1*, a condition that similarly applies to *Unblock Claim 3* and *Block Claim 3*. Furthermore, the duplicate activity *Block Claim 3* cannot be properly represented by any of the selected discovery algorithms, as they do not support duplicate activities in process modeling. Placing *Block Claim 3* inside the payment loop is also incorrect, however, this structure enables the model to incorporate the unblocking mechanism, as well as the withdrawal and repayment of benefits. Additionally, the model permits *Repayment* without a preceding *Withdraw Claim*, highlighting IMr's difficulty in enforcing strict behavioral dependencies between activities. This limitation stems from the recursive nature of the algorithm and the independent handling of subtasks, which lack proper synchronization.

Real-life event logs without domain knowledge

For the other publicly available event logs introduced in Section 7.1, no domain knowledge exists that can be directly encoded as rules to guide process discovery. Instead, we extract declarative constraints from those logs, using the parameter configurations outlined in Section 7.2, to generate our own rules, and then evaluate how incorporating these rules into the IMr discovery technique yields more accurate process models.

Comparison between IMr vs. IMbi. Similar to the UWV case study, we compute the evaluation metrics *dev-count* and *dev-rate* for different process discovery methods and rule sets. The results are presented in the following figures:

- Fig. 12 shows *dev-count*,
- Fig. 13 presents *dev-rate* calculated using event-level rule support as *dev-penalty*,
- Fig. 14 presents *dev-rate* using trace-level rule support as *dev-penalty*.

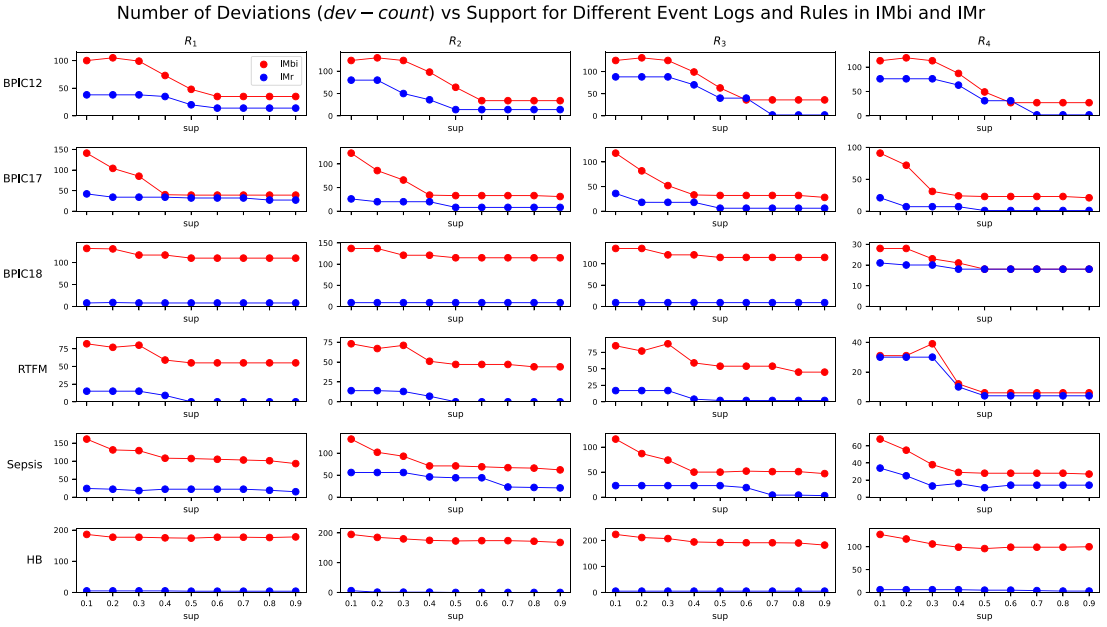


Fig. 12. Comparison of *dev-count* between IMbi (red) and IMr (blue) across different event logs (rows) and rule sets (columns).

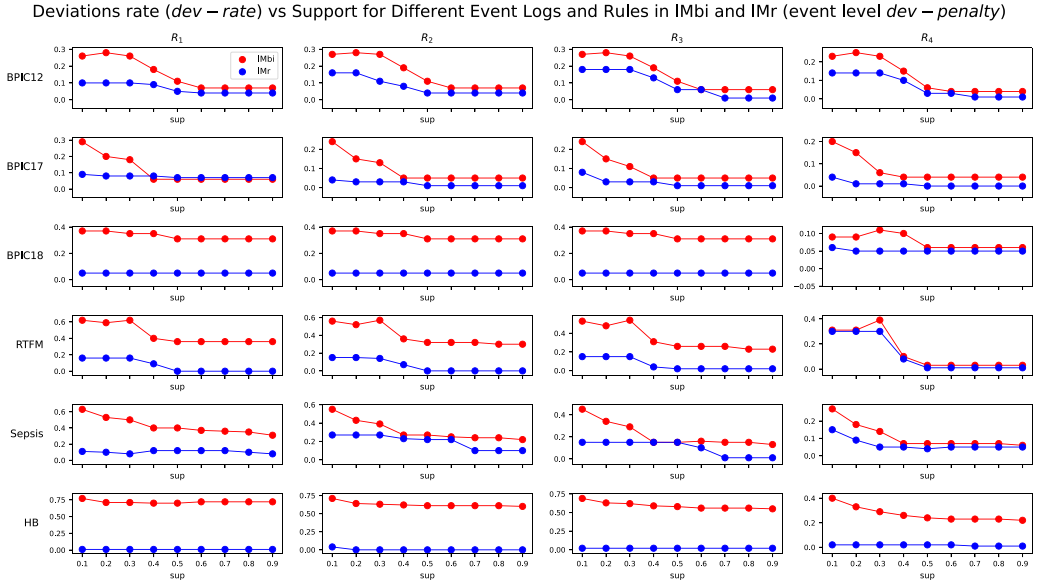


Fig. 13. Comparison of *dev-rate* based on event-level rule support as *dev-penalty*, for IMbi (red) and IMr (blue). Subfigures are arranged by event log (rows) and rule set (columns). (If you are viewing a black-and-white version of this paper, please refer to the online version to interpret the color references in this figure).

In these visualizations, each subfigure corresponds to a specific event log (shown in rows) and rule set (shown in columns). IMbi results are depicted in red, and IMr results in blue. The x-axis of each subfigure represents the *sup* parameter used in the discovery algorithms.

The results demonstrate that the rule sets provided as input are effectively utilized by the IMr technique to discover models that comply with these rules. In contrast, the IMbi technique, which does not incorporate any rules, produces models with notably more deviations. This comparison indicates that IMr successfully integrates the rules into the discovery process, leading to a significant reduction in both the number and severity of deviations.

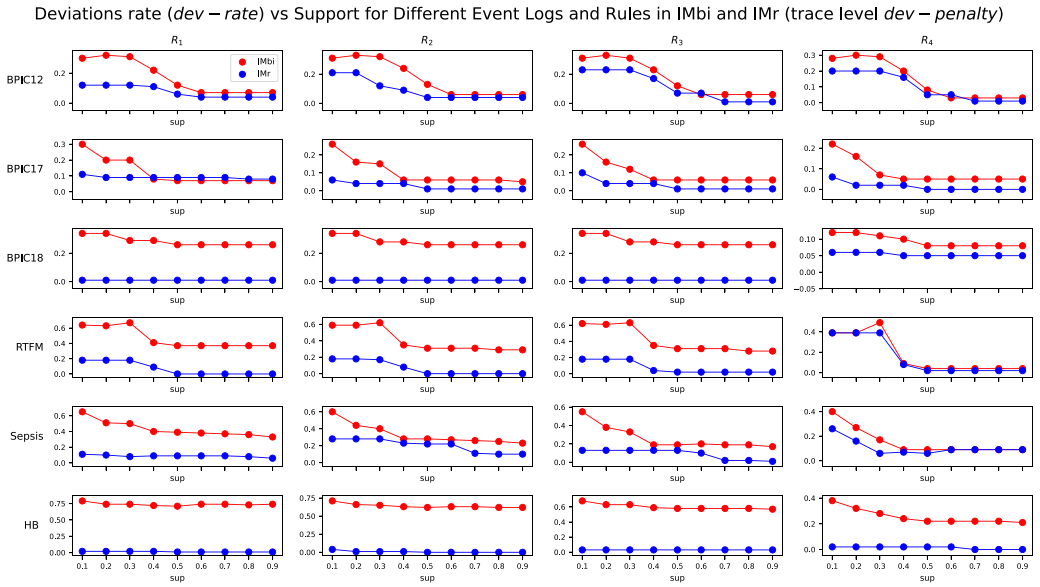


Fig. 14. Comparison of *dev-rate* based on trace-level rule support as *dev-penalty*, for IMbi (red) and IMr (blue). Each subfigure represents a combination of event log (row) and rule set (column). (If you are viewing a black-and-white version of this paper, please refer to the online version to interpret the color references in this figure).

In Fig. 15, fitness, precision, and F1-score are illustrated for different rule configurations. Each subfigure contain the results for a specific event log. The x-axis represents $fitness$, while the y-axis represents $precision$. The dashed lines in the figure correspond to constant $F1\text{-}score$ values, with points lying on the same dashed line sharing the same $F1\text{-}score$. Moving towards the top-right corner of the figure indicates an increase in the $F1\text{-}score$. The different shades of blue represent distinct rule configurations, namely R_1, R_2, R_3, and R_4. In contrast, the red color represents the IMbi-discovered model, which does not use any rules as input. These results highlight that integrating rules as input enhances the quality of the discovered models, as evidenced by noticeable improvements in both $precision$ and $F1\text{-}score$.

Since the effect of parameter settings of the discovery techniques in the quality of the discovered models is less apparent in Fig. 15, we provide a more detailed analysis for selected values of the sup parameter of IMr and IMbi techniques, specifically $sup \in \{0.1, 0.5, 0.9\}$. Table 3 summarizes the conformance checking metrics fitness (fit), precision (prc), and F1-score ($F1$) for each rule set used in the IMr technique as well as the discovered model with IMbi, considering this selection of sup parameter values. In addition, for each rule set, we report the parameter setting that resulted in the best-performing model. The row labeled *best* indicates both the parameter setting (top) and the corresponding conformance checking values (bottom). For each event log, the combination of the parameter setting and rule set that yields the best model is highlighted in bold. If no model achieves a fitness above 90%, there is no best model and therefore, it is indicated with an "X" symbol.

In Fig. 16, we illustrate the fitness, precision, and F1-score for all discovered models across various rule configurations using a box plot. The diagram reveals that the most significant improvement is observed in the precision dimension, which, in turn, leads to an overall increase in the F1-score. This enhancement demonstrates that the rule-guided discovery algorithm effectively steers the process towards more precise process structures, where the behavior allowed by the model aligns more closely with the rules provided as input.

Since the given rules encapsulate additional information about sophisticated relationships between activities, the results highlight that incorporating such rules helps to constrain the behavior permitted by the model, ensuring it does not deviate from the specified rules. Moreover, this additional information refines the model by limiting behaviors not observed in the event log, thereby enhancing its overall quality and adherence to domain-specific requirements.

Comparison to other discovery algorithms. While the primary objective of this paper is to demonstrate the algorithm's effectiveness in satisfying the input rules, it is also important to assess the quality of the discovered models in comparison to state-of-the-art process discovery algorithms. For this comparison, we use the R_3 configuration ($confidence = 0.99$ and $support = 0.005$) for all IMr discoveries. This configuration is chosen because the high $confidence$ value indicates that the activation of rules reliably leads to their satisfaction, while the low $support$ value ensures that the rules are satisfied at least a few times.

A summary of the results is presented in Fig. 17. In this figure, different colors represent various discovery techniques. Specifically, we compared the models discovered using the IMr algorithm with those generated by IMbi, IMf, and split miner, employing the parameter configurations detailed in Section 7.2. The results demonstrate that, even though the input rules are not explicitly designed to optimize the process models, a carefully chosen set of rules can yield models that are comparable to or, in some cases, outperform those generated by state-of-the-art techniques.

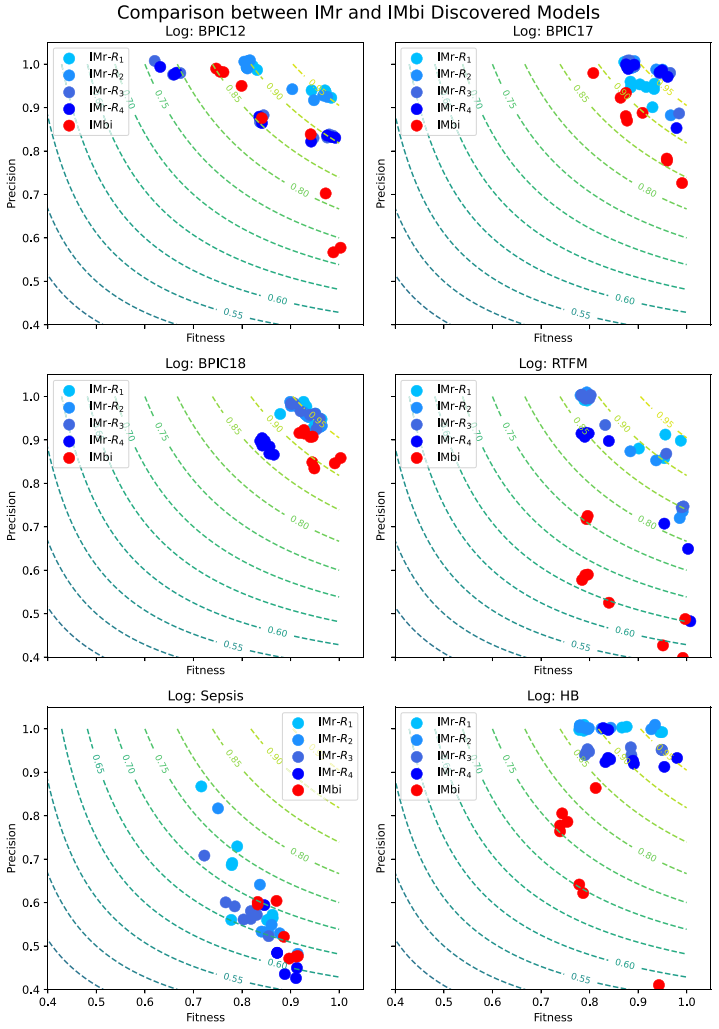


Fig. 15. The comparison between conformance checking results for the IMr discovery technique considering different rule sets (R_1, R_2, R_3, and R_4) and the IMbi discovery technique. (If you are viewing a black-and-white version of this paper, please refer to the online version to interpret the color references in this figure).

Since the correspondence between the parameter settings of each individual process discovery technique and the points represented in Fig. 17 is not visually identifiable, we summarize the conformance checking results for a selection of parameter configurations in Table 4. The selected parameter settings include $sup \in \{0.1, 0.5, 0.9\}$ for IMr and IMbi, $f \in \{0.1, 0.5, 0.9\}$ for IMf, and $(\epsilon, \eta) \in \{(0.1, 0.1), (0.1, 0.5), (0.2, 0.1), (0.2, 0.5)\}$ for split miner. In addition, for each discovery technique, we report the best model that achieved the highest F1-score, provided that its fitness exceeds 90%. These results are shown in the *best* row for each technique. If there is no such model with fitness above 90%, we show it with "X" symbol. For the IMr technique, we report two sets of results: (1) the best model discovered using the R_3 rule set, and (2) the best model across all rule sets (R_1, R_2, R_3, and R_4), shown in the *best-all* row. For each discovery technique and event log, the parameter setting that achieved the best model is displayed on the top row, and the corresponding conformance checking metrics, fitness, precision, and F1-score are shown on the bottom row.

Runtime analysis. The IMbi algorithm requires the exploration of a potentially large set of candidate cuts in each recursion, calculating the cost for each candidate to compare and select the one with the minimum cost (cf. Algorithm 1 with $R = \emptyset$). When rules are provided as input, they are used to prune the set of candidate cuts (cf. Definition 13), effectively reducing the search space. Consequently, this pruning mechanism is expected to lower the computational cost of the algorithm. In Fig. 18, we compare the runtime of the IMbi and IMr algorithms with state-of-the-art process discovery methods, demonstrating the efficiency gains achieved through rule-based pruning.

The IMbi algorithm exhibits the highest computational cost due to the exploration of all candidate cuts and the necessity of calculating the cost for each cut to enable comparisons. When a set of rules is provided as input, the computational cost is significantly reduced in all experiments, as the number of candidate cuts is constrained. Split miner consistently achieves the

Table 3
Conformance checking results (fitness, precision, and F1-score) for IMr and IMbi models at selected *sup* values (0.1, 0.5, 0.9). For each event log and rule set, the best-performing configuration is highlighted in bold. Configurations with fitness below 90% are marked with "X" to indicate the absence of an acceptable model.

	param.	BPIC12			BPIC17			BPIC18			RTFM			Sepsis			HB		
		fit	prc	F1	fit	prc	F1	fit	prc	F1	fit	prc	F1	fit	prc	F1	fit	prc	F1
IMr with R_1	0.1	97	93	95	94	91	92	88	96	92	98	89	93	78	57	66	94	100	97
	0.5	84	99	91	90	95	92	94	96	95	79	100	88	85	57	68	79	100	88
	0.9	81	100	90	88	100	94	93	98	95	79	100	88	72	88	79	79	100	88
	Best	**97**	**0.1** **93**	**95**	93	0.2 95	94	**94**	**0.5** **96**	**95**	**98**	**0.1** **89**	**93**	X	X	X	**94**	**0.1** **100**	**97**
IMr with R_2	0.1	98	92	95	97	89	93	96	93	94	99	73	84	93	38	54	93	100	96
	0.5	81	100	90	88	100	94	94	96	95	79	100	88	85	53	66	79	100	88
	0.9	81	100	90	88	100	94	91	98	94	79	100	88	75	81	78	79	100	88
	Best	**98**	**0.1** **92**	**95**	**96**	**0.2** **98**	**97**	**94**	**0.4** **95**	**95**	95	0.3 86	90	**92**	**0.2** **48**	**63**	93	0.1 100	96
IMr with R_3	0.1	98	83	90	98	89	93	96	93	94	99	74	85	85	51	64	96	95	96
	0.5	84	87	86	88	100	94	94	96	95	79	100	88	81	57	67	80	94	86
	0.9	63	100	77	88	100	94	91	98	94	79	100	88	73	71	72	80	94	86
	Best	98	0.1 83	90	**96**	**0.3** **98**	**97**	**94**	**0.4** **95**	**95**	95	0.3 87	91	X	X	X	96	0.1 95	96
IMr with R_4	0.1	98	83	90	98	85	91	86	87	87	100	49	66	99	29	45	98	93	95
	0.5	84	87	86	88	99	94	84	89	87	79	91	85	91	44	59	83	92	88
	0.9	63	100	77	88	99	94	83	90	86	79	92	85	85	59	70	83	100	91
	Best	98	0.1 83	90	**96**	**0.2** **98**	**97**	X	X	X	95	0.3 72	82	91	0.5 44	59	98	0.1 93	95
IMbi	0.1	100	56	72	99	74	84	100	85	92	100	39	56	99	30	45	95	34	50
	0.5	84	88	86	88	88	88	94	90	92	79	59	67	91	46	61	74	76	75
	0.9	75	98	85	80	99	89	93	92	92	79	72	75	84	60	70	81	87	84
	Best	94	0.2 84	89	90	0.4 89	89	93	0.7 92	92	100	0.2 48	65	**92**	**0.3** **48**	**63**	94	0.1 40	56

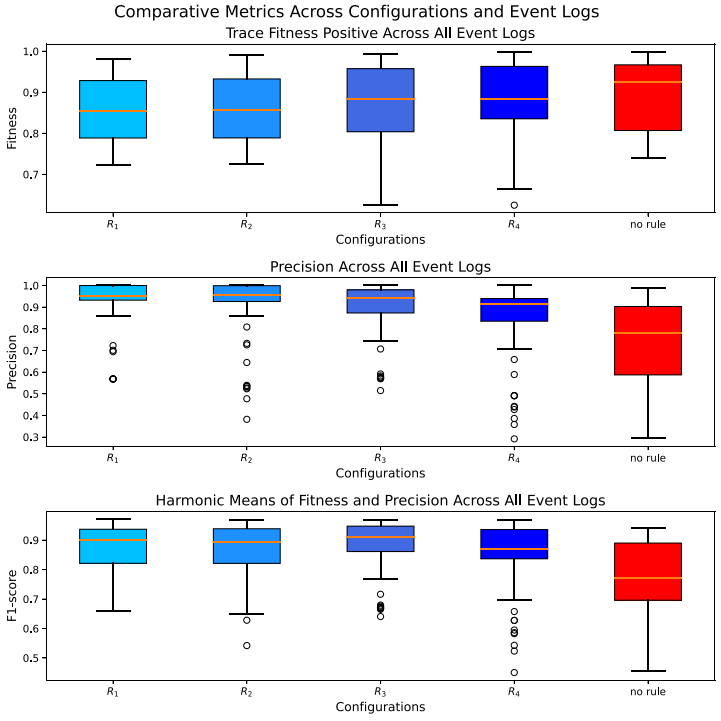


Fig. 16. Box plot showing fitness, precision, and F1-score for all discovered models across different rule configurations. The results highlight significant improvements in precision, leading to an overall increase in the F1-score.

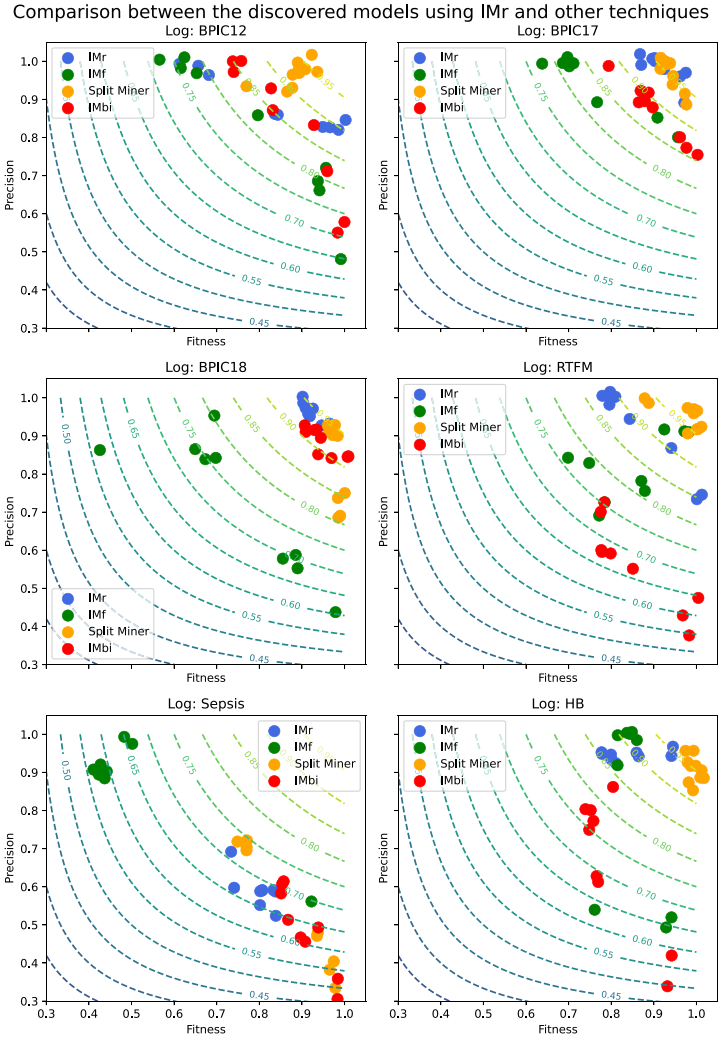


Fig. 17. Comparison of model quality (fitness, precision, and F1-score) between IMr (using R_3 configuration) and state-of-the-art process discovery algorithms. The results show that a well-chosen set of rules can produce models comparable to or superior to those generated by state-of-the-art techniques. (If you are viewing a black-and-white version of this paper, please refer to the online version to interpret the color references in this figure).

lowest computation time across all experiments. Additionally, in certain cases, the IMr algorithm demonstrates better computational performance compared to the IMf method.

The time required to discover declarative constraints from the event logs is not included in Fig. 18. However, extracting such rules may introduce additional computational overhead. In Fig. 19, we present a runtime comparison similar to Fig. 18, but this time incorporating the time needed to discover constraints using the Minerful technique into the total runtime of the IMr approach. The results show that when rule discovery is performed prior to applying IMr, the added overhead may result in minor to no improvement in comparison to IMbi such as for the BPIC12, BPI18, and RTFM logs. However, for more complex event logs such as Sepsis, HB, and UWV, the IMr technique still demonstrates a clear runtime advantage over IMbi, even when including the time required for rule discovery. This is likely due to the higher structural complexity or the larger number of activities in these logs.

Analysis of the discovered models for BPIC2017. To conclude the experimental evaluation, we present selected discovered models for one of the event logs, namely BPIC17, to offer insights into their structure and support a more detailed assessment.

The best models discovered for the BPIC2017 event log are as follows:

– IMr: The discovered model with rule set R_2 and parameter $sup = 0.2$ is the best with $fitness$ of 0.96 and $precision$ of 0.98 and $F1\text{-}score$ of 0.97 (rule set R_3 with $sup = 0.3$ and rule set R_4 with $sup = 0.2$ resulted in the same discovered models).

Table 4
Conformance checking results (fitness, precision, and F1-score) for selected parameter configurations of each process discovery technique. For each technique, the best-performing model is reported in the *best* row. For IMr, both the best model using R_3 and the best model across all rule sets are included (labeled *best-*R_3 and *best-all*, respectively). Configurations with fitness below 90% are marked with "X".

	param.	BPIC12			BPIC17			BPIC18			RTFM			Sepsis			HB		
		fit	prc	F1	fit	prc	F1	fit	prc	F1	fit	prc	F1	fit	prc	F1	fit	prc	F1
IMr	0.1	98	83	90	98	89	93	96	93	94	99	74	85	85	51	64	96	95	96
	0.5	84	87	86	88	100	94	94	96	95	79	100	88	81	57	67	80	94	86
	0.9	63	100	77	88	100	94	91	98	94	79	100	88	73	71	72	80	94	86
	Best-R_3	98	0.1 83	90	96	0.3 98	97	94	0.4 95	95	95	0.3 87	91	X	X X	X	96	0.1 95	96
	Best-all	97	0.1, R_1 93	95	96	0.3, R_3 98	97	94	0.4, R_3 95	95	98	0.1, R_1 89	93	92	0.2, R_2 48	63	94	0.1, R_1 100	97
IMbi	0.1	100	56	72	99	74	84	100	85	92	100	39	56	99	30	45	95	34	50
	0.5	84	88	86	88	88	88	94	90	92	79	59	67	91	46	61	74	76	75
	0.9	75	98	85	80	99	89	93	92	92	79	72	75	84	60	70	81	87	84
	Best	94	0.2 84	89	90	0.4 89	89	93	0.7 92	92	100	0.2 48	65	92	0.3 48	63	94	0.2 40	56
IMf	0.1	97	47	64	96	79	87	97	42	59	98	90	94	95	30	46	93	51	66
	0.5	80	88	84	70	100	82	69	85	76	78	71	75	43	90	58	83	92	87
	0.9	58	100	74	64	100	78	44	87	58	71	84	77	43	90	58	83	99	90
	Best	96	0.2 73	83	91	0.2 85	88	97	0.1 42	59	98	0.1 90	94	90	0.2 56	69	93	0.1 51	66
SM	0.1, 0.1	87	91	89	98	90	94	100	70	82	100	92	96	98	35	52	100	87	93
	0.1, 0.5	91	100	95	93	99	96	98	91	94	89	100	94	76	71	73	99	94	96
	0.2, 0.1	75	92	83	98	90	94	100	70	82	100	92	96	98	35	52	100	86	93
	0.2, 0.5	89	98	93	93	99	96	98	91	94	89	100	94	76	71	73	99	94	96
	Best	92	0.1, 0.3 98	95	93	0.2, 0.5 99	96	98	0.2, 0.5 91	94	100	0.2, 0.3 96	98	94	0.2, 0.3 46	62	99	0.2, 0.5 94	96

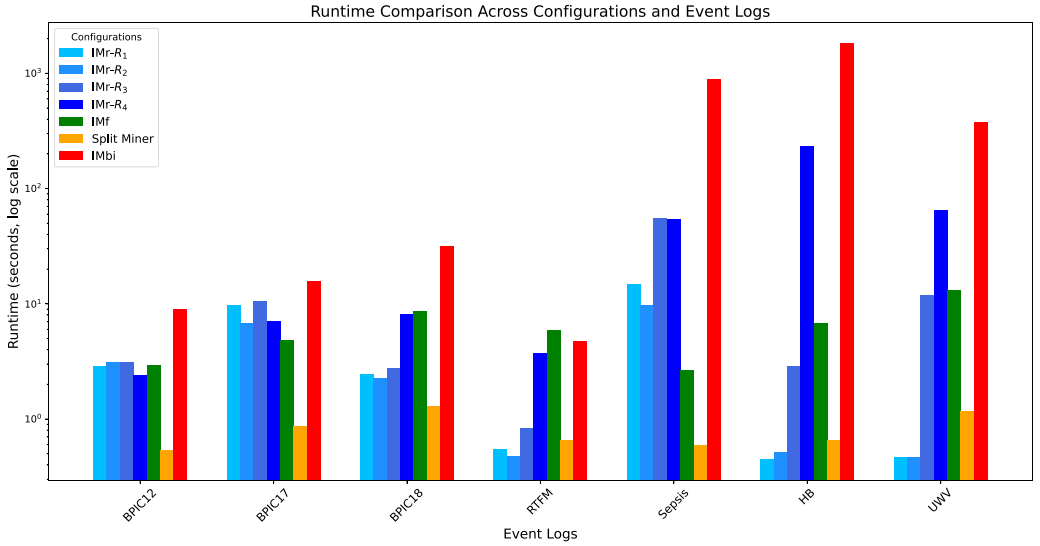


Fig. 18. Runtime comparison of the IMbi, IMr, and IMf algorithms against state-of-the-art process discovery methods. The results show that IMbi has the highest computational cost due to exhaustive exploration, while the use of rules (IMr) reduces computation time by limiting candidate cuts. Split Miner consistently achieves the lowest runtime, with IMr occasionally outperforming IMf in specific cases.

- IMbi: The best model is discovered with the parameter of $sup = 0.4$, this model has a $fitness$ of 0.9 and $precision$ of 0.88 and $F1\text{-}score$ of 0.89.
- IMf: The best model is discovered with parameter $f = 0.2$, this model has a $fitness$ of 0.91, $precision$ of 0.85 and $F1\text{-}score$ of 0.88.
- Split Miner: The best model discovered by the split miner is achieved with $\epsilon = 0.2$ and $\eta = 0.5$. This model has a $fitness$ of 0.93 and $precision$ of 0.99 and $F1\text{-}score$ of 0.95.

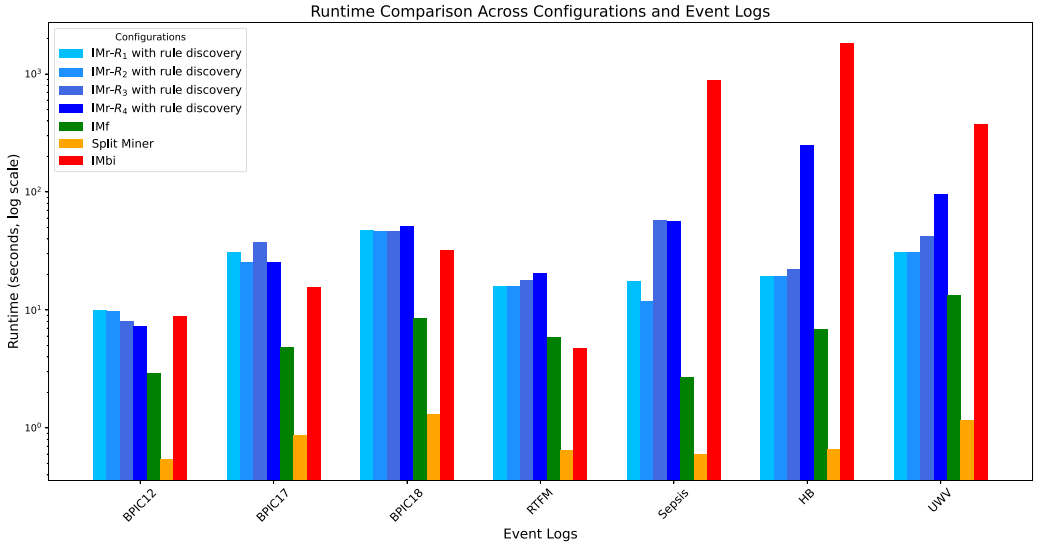
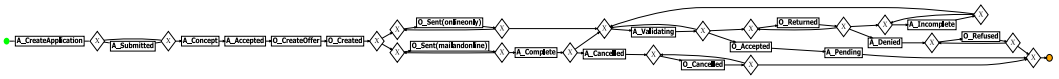


Fig. 19. Runtime comparison of process discovery techniques, including the additional time required for rule discovery using Minerful in the IMr approach.

Table 5
Comparison of metrics across different discovery techniques for the BPIC17 event log.

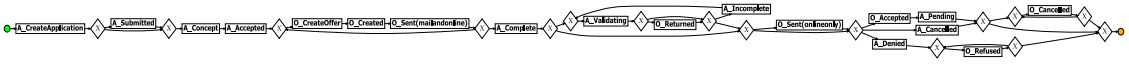
Discovery technique	IMr	IMbi	IMf	Split miner
Parameter setting	$sup = 0.2$	$sup = 0.4$	$f = 0.2$	$\epsilon = 0.2, \eta = 0.5$
fitness	0.96	0.90	0.91	0.93
precision	0.98	0.88	0.85	0.99
F1-score	0.97	0.89	0.88	0.95
dev-count	20	34	26	48
dev-rate	0.03	0.05	0.04	0.11



(a) Process model discovered using split miner with $\epsilon = 0.2$ and $\eta = 0.5$.

(b) Process model discovered using IMf with *sup*=0.2.

(c) Process model discovered using IMbi with *sup*=0.4.



(d) Process model discovered using IMr with *sup*=0.2 and the rule set R_2.

Fig. 20. The best discovered models from BPIC17 event log using different process discovery algorithms.

The conformance checking results (*fitness*, *precision*, and *F1-score*) along with the deviation metrics (*dev-count*, and *dev-rate*) are summarized in Table 5. The corresponding best models are visualized as BPMN models in Fig. 20.

Considering the R_2 rule set, which yielded the best model for IMr, we highlight several rules that deviate in other process discovery methods but are satisfied in the model discovered using the IMr framework represented in Fig. 20(d). These rules are listed below (s represents the abbreviation for *support* and c denotes the abbreviation for *confidence*):

– The discovered model with IMbi represented in Fig. 20(c):

- $AlternateSuccession(A\_Concept, A\_Complete)$, $s = 0.14$, $c = 1.0$
- $NotChainSuccession(A\_Complete, O\_Returned)$, $s = 0.13$, $c = 1.0$
- $AtMost1(A\_Complete)$, $s = 0.07$, $c = 1.0$
- $NotCoExistence(O\_Cancelled, A\_Denied)$, $s = 0.06$, $c = 0.98$
- $NotCoExistence(A\_Pending, A\_Denied)$, $s = 0.05$, $c = 1.0$
- $NotCoExistence(A\_Denied, O\_Accepted)$, $s = 0.05$, $c = 1.0$

– The discovered model with IMf represented in Fig. 20(b):

- $NotChainSuccession(A\_Complete, O\_Returned)$, $s = 0.13$, $c = 1.0$
- $NotChainSuccession(O\_Cancelled, A\_Denied)$, $s = 0.06$, $c = 1.0$
- $NotCoExistence(A\_Pending, A\_Denied)$, $s = 0.05$, $c = 1.0$
- $NotCoExistence(O\_Accepted, O\_Refused)$, $s = 0.05$, $c = 1.0$
- $NotCoExistence(A\_Denied, O\_Accepted)$, $s = 0.05$, $c = 1.0$

– The discovered model with the split miner represented in Fig. 20(a):

- $CoExistence(O\_Sent(mail\ and\ online), O\_Created)$, $s = 0.19$, $c = 0.99$
- $NotChainSuccession(O\_Sent(mail\ and\ online), O\_Created)$, $s = 0.19$, $c = 1.0$
- $CoExistence(O\_Create\ Offer, O\_Sent(mail\ and\ online))$, $s = 0.19$, $c = 0.99$
- $CoExistence(O\_Created, A\_Complete)$, $s = 0.17$, $c = 1.0$
- $CoExistence(O\_Create\ Offer, A\_Complete)$, $s = 0.17$, $c = 1.0$
- $Succession(A\_Accepted, O\_Sent(mail\ and\ online))$, $s = 0.16$, $c = 0.99$

7.4. Discussion

As discussed in Section 5.4 and demonstrated in the experiments, it is infeasible to guarantee full satisfaction of the input rules in the final discovered models. The inductive mining-based approach is inherently biased toward discovering block-structured process trees through a recursive, divide-and-conquer strategy. This recursive decomposition makes it difficult to evaluate constraints whose activation and target activities are distributed across different subtrees, thus leading to potential rule violations in the final model. The representational bias of the inductive miner technique is an inherent limitation. Therefore, extending the idea of guided process discovery with rules to other discovery techniques that support more flexible structures could be a promising direction for future research.

In cases where no candidate cut fully satisfies all input rules, our framework proceeds with the cut that yields the minimum deviation, as defined in Definition 13. This means that some rule violations are accepted, but any selected cut must still be supported by the behavior observed in the event log. An alternative and promising direction could involve assigning priorities to the rules, enabling the enforcement of high-priority constraints even if it requires selecting a cut that is not among the initially identified candidate cuts.

The automatic discovery of declarative rules requires careful selection of parameters for rule evaluation. In our experiments, we tested four different configurations, however, the results did not yield a consistent guideline for identifying the most effective parameter setting. This challenge is further compounded by the lack of domain knowledge for the publicly available real-life event logs, making it difficult to assess which discovered models align more closely with domain expectations.

8. Conclusion

In this paper, we have extended the IMr framework for rule-guided process discovery, addressing several key limitations of the original approach and broadening its applicability. By incorporating user-defined or discovered declarative rules into the process discovery workflow, our framework enables the creation of process models that better align with domain-specific constraints. We provided a mathematical foundation for the cut search algorithm, allowing for the integration of a wider range of rules. Furthermore, we introduced evaluation metrics to quantify rule deviations, offering a clearer assessment of model quality.

Our work also addresses a critical limitation of the original algorithm, where rule conflicts in intermediate recursions led to the exclusion of rules entirely. By enhancing the algorithm to select the most promising cut even in the absence of rule-compliant candidates, we improved its robustness and practical utility. The proposed framework was evaluated using real-world event logs, demonstrating that the proposed discovery algorithm is able to change the process structure in accordance to the rules. We also showed that the conformance quality of the discovered models are comparable with the state-of-the-art discovery techniques.

These results highlight the value of incorporating domain knowledge into process discovery to generate more accurate and interpretable models. The contributions of this work provide both theoretical advancements and practical tools for researchers and practitioners. Future research could explore the scalability of the approach for larger event logs, the integration of additional rule types, and automated methods for assigning rule importance weights. By addressing these directions, the field of rule-guided process discovery can further advance towards more effective and adaptable frameworks.

CRediT authorship contribution statement

Ali Norouzifar: Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Marcus Dees:** Writing – review & editing, Validation, Data curation. **Wil van der Aalst:** Writing – review & editing, Validation, Supervision, Formal analysis.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ali Norouzifar reports financial support was provided by German federal state of North Rhine-Westphalia. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

References

[1] A. Norouzifar, M. Dees, W.M.P. van der Aalst, Imposing rules in process discovery: An inductive mining approach, in: J. ao Araújo, J.L. de la Vara, M.Y. Santos, S. Assar (Eds.), Research Challenges in Information Science - 18th International Conference, RCIS 2024, GuimarÃes, Portugal, May 14-17, 2024, Proceedings, Part I, in: Lecture Notes in Business Information Processing, vol. 513, Springer, 2024, pp. 220–236, http://dx.doi.org/10.1007/978-3-031-59465-6_14.

[2] A. Augusto, R. Conforti, M. Dumas, M.L. Rosa, F.M. Maggi, A. Marrella, M. Mecella, A. Soo, Automated discovery of process models from event logs: Review and benchmark, IEEE Trans. Knowl. Data Eng. 31 (4) (2019) 686–705, http://dx.doi.org/10.1109/TKDE.2018.2841877.

[3] J. Carmona, B.F. van Dongen, A. Solti, M. Weidlich, Conformance Checking - Relating Processes and Models, Springer, 2018.

[4] J.D. Weerdt, S.K.L.M. vanden Broucke, J. Vanthienen, B. Baesens, Active trace clustering for improved process discovery, IEEE Trans. Knowl. Data Eng. 25 (12) (2013) 2708–2720, http://dx.doi.org/10.1109/TKDE.2013.64.

[5] D. Chapela-Campa, M. Mucientes, M. Lama, Understanding complex process models by abstracting infrequent behavior, Future Gener. Comput. Syst. 113 (2020) 428–440, http://dx.doi.org/10.1016/J.FUTURE.2020.07.030.

[6] M.L. van Eck, X. Lu, S.J.J. Leemans, W.M.P. van der Aalst, PM2 : A process mining project methodology, in: J. Zdravkovic, M. Kirikova, P. Johannesson (Eds.), Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings, in: Lecture Notes in Computer Science, vol. 9097, Springer, 2015, pp. 297–313, http://dx.doi.org/10.1007/978-3-319-19069-3_19.

[7] D. Schuster, S.J. van Zelst, W.M.P. van der Aalst, Utilizing domain knowledge in data-driven process discovery: A literature review, Comput. Ind. 137 (2022) 103612, http://dx.doi.org/10.1016/J.COMPIND.2022.103612.

[8] A. Norouzifar, W.M.P. van der Aalst, Discovering process models that support desired behavior and avoid undesired behavior, in: J. Hong, M. Lanperne, J.W. Park, T. Cerný, H. Shahriar (Eds.), Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, SAC 2023, Tallinn, Estonia, March 27-31, 2023, ACM, 2023, pp. 365–368, http://dx.doi.org/10.1145/3555776.3577818.

[9] S. Goedertier, D. Martens, J. Vanthienen, B. Baesens, Robust process discovery with artificial negative events, J. Mach. Learn. Res. 10 (2009) 1305–1340, http://dx.doi.org/10.5555/1577069.1577113, URL https://dl.acm.org/doi/10.5555/1577069.1577113.

[10] T. Slaats, S. Debois, C.O. Back, Weighing the pros and cons: Process discovery with negative examples, in: A. Polyvyanyy, M.T. Wynn, A.V. Looy, M. Reichert (Eds.), Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings, in: Lecture Notes in Computer Science, vol. 12875, Springer, 2021, pp. 47–64, http://dx.doi.org/10.1007/978-3-030-85469-0_6.

[11] F. Chesani, C. Di Francescomarino, C. Ghidini, G. Grundler, D. Loreti, F.M. Maggi, P. Mello, M. Montali, S. Tessaris, Shape your process: Discovering declarative business processes from positive and negative traces taking into account user preferences, in: J. ao Paulo A. Almeida, D. Karastoyanova, G. Guizzardi, M. Montali, F.M. Maggi, C.M. Fonseca (Eds.), Enterprise Design, Operations, and Computing - 26th International Conference, EDOC 2022, Bozen-Bolzano, Italy, October 3-7, 2022, Proceedings, in: Lecture Notes in Computer Science, vol. 13585, Springer, 2022, pp. 217–234, http://dx.doi.org/10.1007/978-3-031-17604-3_13.

[12] A. Norouzifar, H. Kourani, M. Dees, W.M.P. van der Aalst, Bridging domain knowledge and process discovery using large language models, 2024, http://dx.doi.org/10.48550/ARXIV.2408.17316, CoRR abs/2408.17316. arXiv:2408.17316.

[13] D. Fahland, W.M.P. van der Aalst, Repairing process models to reflect reality, in: A. Barros, A. Gal, E. Kindler (Eds.), Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings, in: Lecture Notes in Computer Science, vol. 7481, Springer, 2012, pp. 229–245, http://dx.doi.org/10.1007/978-3-642-32885-5_19.

[14] D. Schuster, S.J. van Zelst, W.M.P. van der Aalst, Cortado: A dedicated process mining tool for interactive process discovery, SoftwareX 22 (2023) 101373, http://dx.doi.org/10.1016/J.SOFTX.2023.101373.

[15] M. Dees, M. de Leoni, F. Mannhardt, Enhancing process models to improve business performance: A methodology and case studies, in: H. Panetto, C. Debruyne, W. Gaaloul, M.P. Papazoglou, A. Paschke, C.A. Ardagna, R. Meersman (Eds.), On the Move To Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I, in: Lecture Notes in Computer Science, vol. 10573, Springer, 2017, pp. 232–251, http://dx.doi.org/10.1007/978-3-319-69462-7_15.

[16] F.M. Maggi, R.P.J.C. Bose, W.M.P. van der Aalst, Efficient discovery of understandable declarative process models from event logs, in: J. Ralyté, X. Franch, S. Brinkkemper, S. Wrycza (Eds.), Advanced Information Systems Engineering - 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings, in: Lecture Notes in Computer Science, vol. 7328, Springer, 2012, pp. 270–285, http://dx.doi.org/10.1007/978-3-642-31095-9_18.

[17] C.D. Ciccio, M. Mecella, On the discovery of declarative control flows for artful processes, ACM Trans. Manag. Inf. Syst. 5 (4) (2015) 24:1–24:37, http://dx.doi.org/10.1145/2629447.

[18] D. Knuplesch, M. Reichert, L.T. Ly, A. Kumar, S. Rinderle-Ma, Visual modeling of business process compliance rules with the support of multiple perspectives, in: W. Ng, V.C. Storey, J. Trujillo (Eds.), Conceptual Modeling - 32th International Conference, ER 2013, Hong-Kong, China, November 11-13, 2013. Proceedings, in: Lecture Notes in Computer Science, vol. 8217, Springer, 2013, pp. 106–120, http://dx.doi.org/10.1007/978-3-642-41924-9_10.

[19] P.M. Dixit, J.C.A.M. Buijs, W.M.P. van der Aalst, B. Hompes, H. Buurman, Enhancing process mining results using domain knowledge, in: P. Ceravolo, S. Rinderle-Ma (Eds.), Proceedings of the 5th International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA 2015), Vienna, Austria, December 9-11, 2015, in: CEUR Workshop Proceedings, vol. 1527, CEUR-WS.org, 2015, pp. 79–94, https://ceur-ws.org/Vol-1527/paper6.pdf.

[20] S.J.J. Leemans, D. Fahland, W.M.P. van der Aalst, Discovering block-structured process models from event logs containing infrequent behaviour, in: N. Lohmann, M. Song, P. Wohed (Eds.), Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, in: Lecture Notes in Business Information Processing, vol. 171, Springer, 2013, pp. 66–78, http://dx.doi.org/10.1007/978-3-319-06257-0_6.
[21] J.N. van Detten, P. Schumacher, S.J.J. Leemans, An approximate inductive miner, in: 5th International Conference on Process Mining, ICPM 2023, Rome, Italy, October 23-27, 2023, IEEE, 2023, pp. 129–136, http://dx.doi.org/10.1109/ICPM60904.2023.10271971.
[22] D. Brons, R. Scheepens, D. Fahland, Striking a new balance in accuracy and simplicity with the probabilistic inductive miner, in: C.D. Ciccio, C.D. Francescomarino, P. Soffer (Eds.), 3rd International Conference on Process Mining, ICPM 2021, Eindhoven, the Netherlands, October 31 - Nov. 4, 2021, IEEE, 2021, pp. 32–39, http://dx.doi.org/10.1109/ICPM53251.2021.9576864.
[23] A. Augusto, R. Conforti, M. Dumas, M.L. Rosa, A. Polyvyanyy, Split miner: Automated discovery of accurate and simple business process models from event logs, Knowl. Inf. Syst. 59 (2) (2019) 251–284, http://dx.doi.org/10.1007/S10115-018-1214-X.
[24] S. Leemans, Robust process mining with guarantees (Ph.D. thesis), Mathematics and Computer Science, 2017, Proefschrift.
[25] S.J.J. Leemans, D. Fahland, W.M.P. van der Aalst, Discovering block-structured process models from event logs - a constructive approach, in: J.M. Colom, J. Desel (Eds.), Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings, in: Lecture Notes in Computer Science, vol. 7927, Springer, 2013, pp. 311–329, http://dx.doi.org/10.1007/978-3-642-38697-8_17.
[26] S.J.J. Leemans, D. Fahland, W.M.P. van der Aalst, Discovering block-structured process models from incomplete event logs, in: G. Ciardo, E. Kindler (Eds.), Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings, in: Lecture Notes in Computer Science, vol. 8489, Springer, 2014, pp. 91–110, http://dx.doi.org/10.1007/978-3-319-07734-5_6.
[27] C.D. Ciccio, M. Montali, Declarative process specifications: Reasoning, discovery, monitoring, in: W.M.P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, in: Lecture Notes in Business Information Processing, vol. 448, Springer, 2022, pp. 108–152, http://dx.doi.org/10.1007/978-3-031-08848-3_4.
[28] C.D. Ciccio, On the Mining of Artful Processes (Ph.D. thesis), Sapienza University of Rome.
[29] A. Alman, C. Di Ciccio, D. Haas, F.M. Maggi, A. Nolte, Rule mining with RuM, in: ICPM, IEEE, 2020, pp. 121–128.
[30] B. van Dongen, BPI challenge 2012, 2012, http://dx.doi.org/10.4121/UUID:3926DB30-F712-4394-AEBC-75976070E91F, URL https://data.4tu.nl/articles/_/12689204/1.
[31] B. van Dongen, BPI challenge 2017, 2017, http://dx.doi.org/10.4121/UUID:5F3067DF-F10B-45DA-B98B-86AE4C7A310B, URL https://data.4tu.nl/articles/_/12696884/1.
[32] B. van Dongen, F.F. Borchert, BPI challenge 2018, 2018, http://dx.doi.org/10.4121/UUID:3301445F-95E8-4FF0-98A4-901F1F204972, URL https://data.4tu.nl/articles/_/12688355/1.
[33] M.M. de Leoni, F. Mannhardt, Road traffic fine management process, 2015, http://dx.doi.org/10.4121/UUID:270FD440-1057-4FB9-89A9-B699B47990F5, URL https://data.4tu.nl/articles/_/12683249/1.
[34] F. Mannhardt, Sepsis cases - event log, 2016, http://dx.doi.org/10.4121/UUID:915D2BFB-7E84-49AD-A286-DC35F063A460, URL https://data.4tu.nl/articles/_/12707639/1.
[35] F. Mannhardt, Hospital billing - event log, 2017, http://dx.doi.org/10.4121/UUID:76C46B83-C930-4798-A1C9-4BE94DFEB741, URL https://data.4tu.nl/articles/_/12705113/1.

Ali Norouzifar is a PhD student at RWTH Aachen University in the Process and Data Science (PADS) group. He holds both a Master's and a Bachelor's degree in Control Engineering from Isfahan University of Technology. His PhD research is mainly part of the DataNinja Research Training Group, funded by the German federal state of North Rhine-Westphalia. His main research interests include event data analysis and process discovery. In his work, he investigates how the control flow of processes can vary across different perspectives and leverages this knowledge to discover process models that more accurately represent a given perspective.

Marcus Dees has worked at Uitvoeringsinstantie Werknemersverzekeringen (UWV) as Customer Intelligence Analyst. UWV is the Employee Insurance Agency of the Netherlands. Since 2025 Marcus works as an integral capacity management consultant at Radboud university medical center (Radboudumc) in Nijmegen, the Netherlands. In his work Marcus connects present-day business problems with scientific research in the Business Process Management and the Process Mining domain.

Wil 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.