Scalable Discovery of Partially Ordered Workflow Models with Formal Guarantees

Humam Kourani
Fraunhofer FIT
Sankt Augustin, Germany
humam.kourani@fit.fraunhofer.de

Daniel Schuster
Fraunhofer FIT
Sankt Augustin, Germany
daniel.schuster@fit.fraunhofer.de

Wil van der Aalst
RWTH Aachen University
Aachen, Germany
wvdaalst@pads.rwth-aachen.de

***Abstract*—Many real-life processes naturally define partial orders over the activities they are composed of. Partial orders can be used as a graph-like representation of process behavior, allowing us to model concurrent and sequential dependencies. The Partially Ordered Workflow Language (POWL) combines block-structured modeling notations with partially-ordered graph representations. A POWL model is a hierarchical model where sub-models can be combined into a new model either using a control-flow operator or as a partial order. The application of POWL models in process mining remains a challenge due to a lack of scalable approaches for the discovery of POWL models. In this paper, we address this gap by proposing an approach for the discovery of POWL models that leverages large data sets and ensures high conformity with the input data. Our approach provides formal guarantees on the uniqueness, existence, and quality of discovered partial orders. The evaluation of our approach underscores its scalability with large data sets and its ability to generate high-quality models.**

***Index Terms*—partial order, process discovery, inductive mining**

I. Introduction

Process models serve as a representation of processes, facilitating communication, simulation, and analysis. Process models can either be created by hand or discovered using process discovery techniques. Organizations use information systems to track and record data about the execution of their processes, and these data are used for the discovery of process models. Discovered models provide insights that empower organizations to streamline their processes and improve decision-making, leading to enhanced automation and efficiency.

Partial orders are used as a representation of the execution order of activities for many real-life processes. In a partial order, some activities may have a strict order with respect to each other (e.g., activity “a” must happen before activity “b”), while other activities are concurrent (e.g., activities “b” and “c” may happen in any order). Partially ordered graph representations allow us to model concurrency and sequential dependencies in an efficient and compact manner; however, they lack support for cyclic process behavior, which is very common in practice. Moreover, in a partial order over activities, we assume all activities to be executed, and thus, modeling a choice is not supported.

A *process tree* [16] is a block-structured process modeling notation that uses control-flow operators (sequence $\rightarrow$, choice $\times$, concurrency $+$, and loop $\circlearrowleft$) to model behavioral dependencies between different blocks of activities. The Partially Ordered Workflow Language (POWL) [15] is a hybrid modeling language that extends partially ordered graphs with process tree operators for modeling choice and loop structures; i,e, a POWL model is a hierarchical model where sub-models can be combined either as a partial order or using process tree operators.

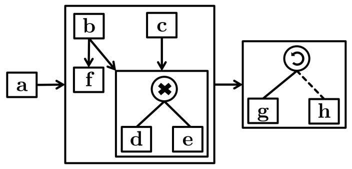

Fig. 1: A POWL model. We only visualize the transitive reduction of a partial order for the sake of simplicity

.

An example POWL model is depicted in Fig. 1. The outer layer of the hierarchy is a partial order modeling a sequence between the activity sets $\{a\}$, $\{b, c, d, e, f\}$, and $\{g, h\}$. Another partial order is used to model non-hierarchical dependencies between $\{b\}$, $\{c\}$, $\{d, e\}$, and $\{f\}$; these dependencies cannot be modeled by a process tree as process trees are limited to hierarchical structures [16]. Finally, the process tree operators $\times$ and $\circlearrowleft$ are used to model the choice between d and e and the loop between g and h respectively.

Process discovery is one of the main branches of process mining. Process discovery techniques analyze event data, aiming at learning a process model that captures the behavior recorded in the data. In [26] the authors outline four levels of maturity for process discovery techniques. In [15], we proposed an initial discovery approach that demonstrates the feasibility of using POWL models for process discovery. This initial approach is not scalable to handle large real-life data sets (i.e., there exists no discovery approach of the second maturity level for POWL models [26]). The absence of a mature, scalable discovery approach underscores the challenge of developing comprehensive POWL models.

In this paper, we contribute towards filling this gap by proposing a scalable approach for the discovery of POWL models. Moreover, the proposed method improves the quality of the discovered models by ensuring a high degree of conformity between the discovered models and the input event data. We achieve this by defining a notion of completeness, which serves as a mechanism to encapsulate all the order restrictions between activities that the data permits. We also define a notion of maximality, which provides formal guarantees on the uniqueness and existence of the discovered partial orders. We believe that our contribution will significantly enhance the robustness and applicability of POWL models in practice.

The remainder of the paper is structured as follows. We discuss related work in Sec. II, and we present basic preliminaries in Sec. III. We introduce our approach for the discovery of POWL models in Sec. IV. Next, we evaluate our approach using real-life event data in Sec. V. Finally, we summarize the paper and outline future work in Sec. VI.

II. Related Work

Various process discovery approaches have been proposed. We refer to [3] for an overview of automated process discovery methods used in the field of process mining. In [16], the inductive mining framework is introduced. The approach we propose for the discovery of POWL models extends the inductive mining framework.

Partial orders are used for data representation and process modeling. An overview of the use of partial orders in process mining is provided in [17]. In [12], the authors propose three approaches for the discovery of Petri nets by aggregating partially ordered sets of events. Another approach for the discovery of Petri nets from partially ordered languages is introduced in [4]. In [21], Mannila et al. propose an approach for the discovery of frequent episodes, where an episode is defined as a partially ordered set of events. In [14], the authors create partially ordered representations of activities and combine them into a workflow graph. In [13], the authors suggest a discovery approach that generates partially ordered graphs enriched with a conflict relation. This approach is able to model choice due to the conflict relation; however, loops remain a major challenge for prime event structures. In [22], the authors present a method for deriving conditional partial order graphs from event logs. A conditional partial order graph [23] is a compact representation of a family of partial orders that is able to model choice structures, but it fails to capture cyclic behavior.

In [15], we introduce POWL models as a novel process modeling notation that extends partially ordered graphs with control-flow operators, and we introduce an initial discovery approach that demonstrates the feasibility of using POWL models for process discovery. This initial approach is not able to scale on large real-life data sets as it mines for partial order using a brute-force technique.

POWL models can be viewed as hybrid process models; i.e., POWL models combine partially ordered graphs with process trees. Many ideas for combining different modeling notations have been proposed. A hybrid Petri net [2] is defined as a Petri net extended with informal arcs connecting transitions. In [25], another type of hybrid process models is defined, combining imperative and declarative modeling languages.

III. Preliminaries

In this section, we present basic preliminaries.

A. Notation

$\mathbb{N} = \{1, 2, 3, ...\}$ denotes the set of natural numbers. For n sets $X_1, ..., X_n$, the *n-ary Cartesian product* is denoted by $X_1 \times ... \times X_n$. An *$n$-ary relation* over $X_1, ..., X_n$ is a subset of the n-ary Cartesian product $X_1 \times ... \times X_n$.

A *multi-set* generalizes the notion of a set and allows for multiple occurrences of the same element. We define a multi-set M over a set X as a function $M \colon X \to \mathbb{N} \cup \{0\}$. We write a multi-set as $M = [x_1{}^{c_1}, ..., x_n{}^{c_n}]$ where $M(x_i) = c_i$ for $1 \leq i \leq n$ (for $x \in X$ with $M(x) = 1$, we omit the superscript; in case $M(x) = 0$, we omit x). We use $\mathcal{M}(X)$ to denote the set of all multi-sets over X.

We define a sequence over a set X as a function $\sigma \colon \{1, \dots, n\} \to X$, and we write $\sigma = \langle \sigma(1), ..., \sigma(n) \rangle$. We use $|\sigma| = n$ to denote the length of σ and X^* to denote the set of all sequences over X.

Let $\prec \subseteq X \times X$ be a binary (i.e., 2-ary) relation over a set X. For $(x_1, x_2) \in X \times X$, we write $x_1 \prec x_2$ to denote that $(x_1, x_2) \in \prec$, and we write $x_1 \not\prec x_2$ to denote that $(x_1, x_2) \notin \prec$. We refer to $\rho = (X, \prec)$ as an *ordered set*. We call $\prec$ a *strict partial order* if it is *irreflexive* (i.e., $x \not\prec x$ for all $x \in X$) and *transitive* (i.e., $x_1 \prec x_2 \wedge x_2 \prec x_3 \Rightarrow x_1 \prec x_3$)[1]. In the remainder of the paper, we use the term *partial order* to refer to a strict partial order. The *transitive reduction* of $\prec$ is defined as $\prec^- = \{(x_1, x_3) \in \prec \mid \nexists x_2 \in X \Big(x_1 \prec x_2 \wedge x_2 \prec x_3\Big)\}$. We refer to $\rho = (X, \prec)$ as a *partially ordered set (poset)*. We use $\Pi(X)$ to denote the set of all posets over X.

B. Event Log

We use Σ to denote the universe of activities. We define an *event log* $L \in \mathcal{M}(\Sigma^*)$ as a multi-set of activity sequences. A *trace* $\sigma \in L$ is a sequence of activities that represents the execution of a single process instance.

Let $L \in \mathcal{M}(\Sigma^*)$ be an event log. $\Sigma_L = \{a \in \Sigma \mid \exists \sigma \in L, 1 \leq i \leq |\sigma| \Big(\sigma(i) = a\Big)\}$ denotes the set of activities that occur in L. We use $L_{\triangleright} = \{a \in \Sigma_L \mid \exists \sigma \in L \Big(\sigma(1) = a\Big)\}$ to denote the set of *start activities* and $L_{\square} = \{a \in \Sigma_L \mid \exists \sigma \in L \Big(\sigma(|\sigma|) = a\Big)\}$ to denote the set of *end activities*. The *Directly-Follows Graph (DFG)* is a 2-ary relation $\mapsto_L \subseteq \Sigma_L \times \Sigma_L$ that captures direct successions between activities: $a \mapsto_L b$ iff $\exists \sigma \in L, 1 \leq i < |\sigma| \Big(\sigma(i) = a \wedge \sigma(i+1) = b\Big)$. The *Eventually-Follows Graph (EFG)* $\rightsquigarrow_L \subseteq \Sigma_L \times \Sigma_L$ captures direct and indirect successions between activities: $a \rightsquigarrow_L b$ iff $\exists \sigma \in L, 1 \leq i < j \leq |\sigma| \Big(\sigma(i) = a \wedge \sigma(j) = b\Big)$. For an activity a, the *pre-set* of a is defined as $\bullet\!\rightsquigarrow_L a = \{b \mid b \rightsquigarrow_L a \wedge a \not\rightsquigarrow_L b\}$, while the *post-set* of a is defined as $a \rightsquigarrow_L\!\bullet = \{b \mid a \rightsquigarrow_L b \;\wedge\; b \not\rightsquigarrow_L a\}$.

For instance, $L_1 = [\langle a, b, c \rangle^3, \langle a, b, d \rangle^2]$ is an event log consisting of five traces with $\Sigma_{L_1} = \{a, b, c, d\}$,

[1]Irreflexivity and transitivity imply *asymmetry* (i.e., $x_1 \prec x_2 \Rightarrow x_2 \not\prec x_1$).

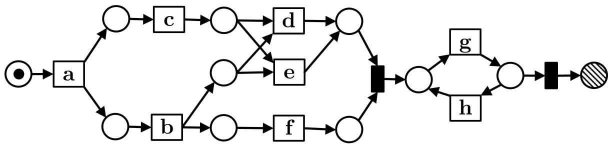

Fig. 2: A sound WF-net modeling the same behavior of the POWL model shown in Fig. 1.

$L_{1\triangleright}=\{a\}$, $L_{1\square}=\{c,d\}$, $\mapsto_{L_1}=\{(a,b),(b,c),(b,d)\}$, and $\rightsquigarrow_{L_1}=\{(a,b),(a,c),(a,d),(b,c),(b,d)\}$. The pre-set of c is $\bullet\!\rightsquigarrow_L c = \{a,b\}$, while the post-set of c is $c\!\rightsquigarrow_L\!\bullet = \emptyset$.

Let $P = \{A_1, ..., A_n\}$ be a partitioning of the activities of an event log L into $n\geq 2$ subsets, i.e., $\Sigma_L=A_1\cup ...\cup A_n$ and $A_i\cap A_j=\emptyset$ for $1\leq i<j\leq n$. For any activity $a \in \Sigma_L$, we write P_a to denote the subset of the partitioning that contains a, i.e., $a \in P_a \in \{A_1, ..., A_n\}$.

C. POWL Model

We use $\tau \notin \Sigma$ to denote the *silent activity* (e.g., τ is used to model a choice between executing some activity or skipping it). A *POWL model* [15] is a partially ordered graph representation of a process, extended with control-flow operators for modeling choice and loop structures. There are three types of POWL models. The first type is the *base case* consisting of a single activity. For the second type, the operators $\times$ and $\circlearrowleft$ can be used to combine multiple POWL models into a new model. The operator $\times$ is used to model an exclusive choice of $n\geq 2$ POWL models, while the operator is used $\circlearrowleft$ to model a do-redo loop of two POWL models. The third type of POWL models is defined as a poset of $n\geq 2$ POWL models. Unconnected nodes in a poset are interpreted to be concurrent, and connections between nodes represent sequential dependencies.

Definition 1 (POWL Model)**.** *A POWL model is recursively defined as follows:*

- *Any activity $a \in \Sigma\cup\{\tau\}$ is a POWL model.*
- *Let ψ_1 and ψ_2 be two POWL models. $\circlearrowleft(\psi_1, \psi_2)$ is a POWL model.*
- *Let $P = \{\psi_1, ..., \psi_n\}$ be a set of $n\geq 2$ POWL models.*
  - *$\times(\psi_1, ..., \psi_n)$ is a POWL model.*
  - *A poset $\rho \in \Pi(P)$ is a POWL model.*

For the formal definition of the semantics of POWL models, we refer to [15]. *Workflow nets* [1] are a powerful modeling notation widely used to formally describe the behavior of business processes. A POWL model can be transformed into a WF-net, and the generated WF-net is guaranteed to be *sound* (i.e., it adheres to certain structural quality criteria) [15]. The POWL model shown in Fig. 1 can be transformed into the sound WF-net shown in Fig. 2.

IV. Discovery of POWL Models with Guarantees

In this section, we propose a scalable approach for the discovery of POWL models. Our approach provides formal guarantees on the uniqueness, existence, and quality of the discovered partial orders.

A. Inductive Miner

The inductive miner [16] is one of the leading approaches in process discovery. It provides formal guarantees such as soundness and rediscoverability of certain process structures. There are several variants of the inductive miner (e.g., for handling incompleteness or infrequent behavior). In this section, we introduce the base variant of the inductive miner that returns a process tree that perfectly fits the input event log (i.e., it discovers a model that covers all behavior recorded in the log). Our approach for the discovery of POWL models adapts the inductive miner to additionally mine for partial orders.

The inductive miner is a recursive top-down approach. The algorithm tries to detect a *cut*, i.e., it tries to detect a behavioral pattern in the directly-follows graph and a partitioning of the activities according to this pattern. The inductive miner supports four cuts corresponding to the four operators of process trees, and it recursively generates a process tree based on the detected cuts.

After detecting a process tree cut, the event log is projected into the different subsets of the partitioning, creating several sub-logs. The same approach is then recursively applied to all sub-logs until a *base case* of the recursion is reached. A base case is defined as an event log whose activity set consists of a single activity. A base case can be easily transformed into a process tree: either into a single node or using the operators $\times$ or $\circlearrowleft$ to model an optional activity or a self-loop.

If neither a base case nor a cut is detected, then the inductive miner invokes a *fall-through function*. This function always returns a cut that might correspond to an underfitting model (i.e., a model that does not precisely capture the behavior recorded in the log), but it allows for continuing the recursion. For the formal description of the different steps of the inductive miner, we refer to [16].

B. Mining for Partial Orders

Fig. 3 illustrates our approach for the discovery of POWL models. It extends the inductive miner to mine for a partial order after mining for the process tree cuts and before invoking the fall-through function. Similarly to process tree cuts, a partial order cut is defined as a partitioning of the activities and a partial order over the partitioning.

Definition 2 (Partial Order Cut)**.** *Let $L \in \mathcal{M}(\Sigma^*)$ be an event log. A partial order cut of L is a poset $(\{A_1, ..., A_n\}, \prec)$ over a partitioning of the activities into $n\geq 2$ subsets $A_1, ..., A_n$.*

We define a notion of *validity* for partial order cuts. For a given partitioning of activities, a *valid* partial order cut is defined as a behavioral pattern in the eventually-follows graphs that corresponds to a partial order over the partitioning of activities [15].

Definition 3 (Valid Partial Order Cut)**.** *Let $L \in \mathcal{M}(\Sigma^*)$ be an event log and $\rho = (P, \prec)$ be a partial order cut of L. ρ*

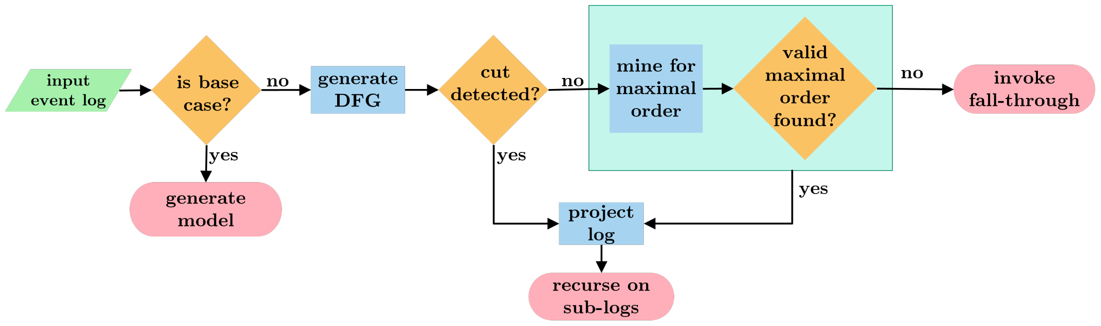

Fig. 3: Approach for the discovery of POWL models.

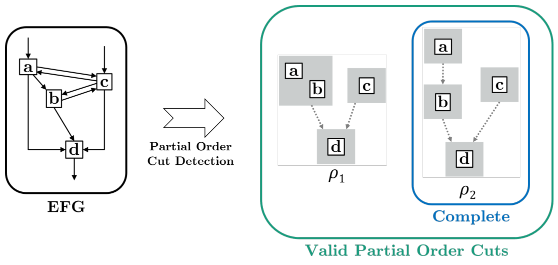

Fig. 4: An eventually-follows graph and two corresponding partial order cuts. The first cut ρ_1 is valid but not complete, while the second cut ρ_2 is valid and complete.

is valid if the following conditions hold for all $A_i, A_j \in P$*;* $A_i{\neq}A_j$*:*

1) $A_i{\prec}A_j$ *iff* $\forall a_i \in A_i, a_j \in A_j \Big(a_i{\rightsquigarrow}_L a_j \wedge a_j{\not\rightsquigarrow}_L a_i\Big)$.
2) $(A_i{\not\prec}A_j \wedge A_j{\not\prec}A_i)$ *iff* $\forall a_i \in A_i, a_j \in A_j \Big(a_i{\rightsquigarrow}_L a_j \wedge a_j{\rightsquigarrow}_L a_i\Big)$.
3) *if* $\nexists A_k \in P\Big(A_k{\prec}A_i\Big)$*, then* $A_i{\cap}L_{\triangleright}{\neq}\emptyset$.
4) *if* $\nexists A_k \in P\Big(A_i{\prec}A_k\Big)$*, then* $A_i{\cap}L_{\square}{\neq}\emptyset$.

After failing to discover a process tree cut, our approach mines for valid partial order cuts. If a valid partial order cut is detected, then the log is projected into the different subsets of the partitioning and the recursion continues; otherwise, the fall-through is invoked. Note that $\rightarrow$ and $+$ cuts are transformed into partial order cuts since POWL models do not support the process tree operators $\rightarrow$ and $+$.

Fig. 4 shows an example eventually-follows graph and two valid partial order cuts detected based on it.

In order to ensure the efficiency of our approach and the uniqueness of the solution, we define a notion of *maximality* for partial order cuts, and we only mine for a maximal partial order cut (cf. Sec. IV-C).

Our approach is *fitness-preserving*, i.e., all traces in the input event log are guaranteed to be included in the language of the generated model. All steps of the inductive miner are fitness-preserving [16], and mining for valid partial order cuts preserves the fitness guarantee as well [15].

C. Maximal Partial Order Cut

We define a notion of maximality for partial order cuts. The goal is to ensure the efficiency of the discovery approach by only mining for a unique solution (i.e., the maximal cut), and at the same time, we need to ensure the optimality of this unique solution. In other words, we mine for a unique partial order cut that not only conforms with the event log but also is the most compact representation of the event log's dependencies, providing existence and validity guarantees if any other solutions exist.

We define a notion of completeness for partial order cuts by exploiting the eventually-follows graph, and then we define maximality over complete partial order cuts. The notion of validity (Def. 3) enforces a certain level of compliance between a valid partial order cut and the EFG; however, it still allows for leaving out some order restrictions derived from the EFG. For example, both partial order cuts shown in Fig. 4 are valid; however, we consider ρ_2 to be better than ρ_1 as ρ_1 does not capture the sequential dependency between the activities a and b.

We define a partial order cut to be *complete* if it fully conforms with the eventually-follows graph. In other words, we ensure capturing all order restrictions derived from the eventually-follows graph in a complete partial order cut, and at the same time, we avoid adding any additional dependencies not observed in the eventually-follows graph.

Definition 4 (Complete Partial Order Cut)**.** *Let* $L \in \mathcal{M}(\Sigma^*)$ *be an event log and* $\rho = (P, \prec)$ *be a partial order cut of* L*.* ρ *is complete if the following conditions hold for all* $A_i, A_j \in P$*;* $a_i \in A_i$*;* $a_j \in A_j$*:*

- $A_i{\prec}A_j$ *iff* $a_i{\rightsquigarrow}_L a_j \wedge a_j{\not\rightsquigarrow}_L a_i$.

In Fig. 4, the first partial order cut is not complete as a and b are in the same subset of the partitioning, i.e., $a{\rightsquigarrow}_L b{\wedge}b{\not\rightsquigarrow}_L a$ and $\{a,b\}{\not\prec}_1\{a,b\}$. The second partial order cut is complete.

We define a complete partial order cut to be *maximal* if the subsets of the partitioning are of maximal size. In other words, a partial order cut is maximal if it is complete and no subsets of the partitioning can be merged without violating the completeness of the order.

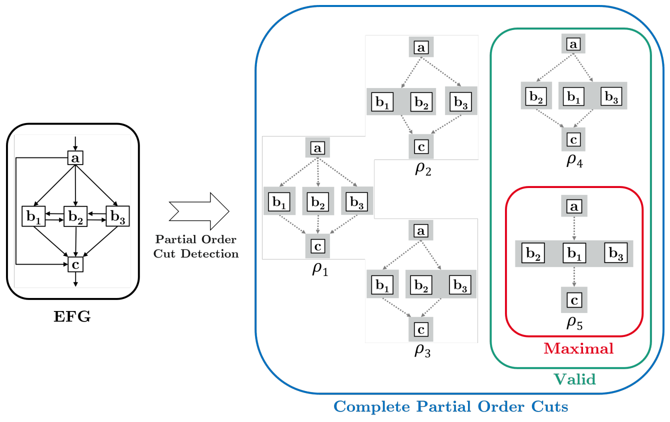

Fig. 5: An eventually-follows graph and five corresponding complete partial order cuts. Since ρ_4 is valid, the validity of the maximal cut ρ_5 is guaranteed (cf. Theorem 3).

Definition 5 (Maximal Partial Order Cut)**.** *Let $L \in \mathcal{M}(\Sigma^*)$ be an event log and $\rho = (P, \prec)$ be a complete partial order cut of L. ρ is maximal if for all $A_i, A_j \in P$; $a_i \in A_i$; $a_j \in A_j$:*

- $A_i = A_j$ *iff* $\bullet{\rightsquigarrow_L}a_i = \bullet{\rightsquigarrow_L}a_j \wedge a_i{\rightsquigarrow_L}\bullet = a_j{\rightsquigarrow_L}\bullet$.

Fig. 5 shows an eventually-follows graph and five partial order cuts detected based on it. All five partial order cuts are complete as they capture the same order restrictions derived from the eventually-follows graph while grouping concurrent activities differently. In this example, ρ_5 is maximal because none of the subsets $\{a\}$, $\{b_1, b_2, b_3\}$, and $\{c\}$ can be merged together. Although all five cuts model the same ordering restrictions between activities, we consider the maximal cut ρ_5 to be the optimal solution due to three reasons. First, the maximal cut is better in terms of simplicity; the graph representation of ρ_5 only has two ordering edges compared to six edges required for representing ρ_1. Second, grouping concurrent activities together into one node allows us to mine for more structures in the next iterations of the discovery approach. In our example, there is a choice between b_1 and b_3. Grouping these activities together into the same node allows us to detect a choice cut in the next iterations. Finally, the maximal cut provides several guarantees:

- If a maximal partial order cut exists, then it is unique (cf. Theorem 1).
- If a complete partial order cut exists, then the maximal partial order cut exists as well (cf. Theorem 2).
- If a complete partial order cut exists and is valid, then the maximal partial order cut exists and is valid as well (cf. Theorem 3).

Next, we prove the three guarantees for maximal partial order cuts. First, we prove the uniqueness guarantee.

Theorem 1 (Uniqueness of Maximal Order)**.** *Let $L \in \mathcal{M}(\Sigma^*)$ be an event log. If a maximal partial order cut of L exists, then it is unique.*

Proof. The maximality condition (Def. 5) uniquely defines a partitioning of activities: two activities a_i and a_j are in the same subset if and only if $\bullet{\rightsquigarrow_L}a_i = \bullet{\rightsquigarrow_L}a_j$ and $a_i{\rightsquigarrow_L}\bullet = a_j{\rightsquigarrow_L}\bullet$. Since activities within the same subset of the partitioning share the same predecessors and successors, the completeness condition (Def. 4) uniquely defines a binary relation $\prec$ over the partitioning: $A_i{\prec}A_j$ iff $\forall\, a_i \in A_i, a_j \in A_j \Big(a_i{\rightsquigarrow_L}a_j \wedge a_j{\not\rightsquigarrow_L}a_i\Big)$.

The uniquely defined binary relation $\prec$ is guaranteed to be irreflexive since it is impossible to fulfill $a{\rightsquigarrow_L}a \wedge a{\not\rightsquigarrow_L}a$ for any activity a. If $\prec$ violates transitivity, then the maximal order does not exist. If $\prec$ is transitive, then $\prec$ is a partial order adhering to the requirements of completeness and maximality by construction, i.e., a maximal order exists and is unique due to the uniqueness of the partitioning and the binary relation. $\square$

Our notion of maximality provides an existence guarantee; i.e., if a complete partial order cut exists, then the maximal partial order cut exists as well.

Theorem 2 (Existence Guarantee of Maximal Order)**.** *Let $L \in \mathcal{M}(\Sigma^*)$ be an event log. If a complete partial order cut of L exists, then a maximal partial order cut of L exists as well.*

Proof. Let $\rho = (P, \prec)$ be the ordered partitioning uniquely defined by Def. 5. In the proof of Theorem 1, we showed that if $\prec$ is transitive, then ρ is the unique maximal order cut of L; and if $\prec$ is not transitive, then no maximal order of L exists.

We prove Theorem 2 by contradiction, i.e., assume that a complete partial order cut $\rho' = (P', \prec')$ of L exists, while $\prec$ is not transitive (i.e., a maximal order cut does not exist).

Since $\prec$ is not transitive, there must exist $a, b, c \in \Sigma_L$ such that $P_a{\prec}P_b$ and $P_b{\prec}P_c$ but $P_a{\not\prec}P_c$. Since $\prec$ satisfies the completeness condition by construction, it follows that $a{\rightsquigarrow_L}b \wedge b{\not\rightsquigarrow_L}a$, $b{\rightsquigarrow_L}c \wedge c{\not\rightsquigarrow_L}b$, and $a{\not\rightsquigarrow_L}c \vee c{\rightsquigarrow_L}a$

The completeness of ρ' implies $P'_a{\prec'}P'_b$, $P'_b{\prec'}P'_c$, and $P'_a{\not\prec'}P'_c$. This means that $\prec'$ is not a partial order as it violates transitivity; therefore ρ' is not a partial order cut. $\square$

Our notion of maximality provides a validity guarantee, i.e., if a valid complete partial order cut exists, the maximal partial order cut is valid as well. We define an auxiliary lemma we use in the proof of the validity guarantee.

Lemma 1 (Inclusion of Partitioning Subsets)**.** *Let $L \in \mathcal{M}(\Sigma^*)$ be an event log and $\rho = (P, \prec)$ be a maximal partial order cut of L. The partitioning subsets of any complete partial order cut of L are subsets of the partitioning subsets of ρ.*

Proof. Let $\rho' = (P', \prec')$ be a complete partial order cut over L. We prove Lemma 1 by contradiction, i.e., assume there are two activities a and b in the same subset in P' (i.e., $P'_a = P'_b$) that they are in separate subsets in P (i.e., $P_a{\neq}P_b$).

Since a and b are in separate subsets in the maximal order, then it follows that either $\bullet{\rightsquigarrow_L}a{\neq}\bullet{\rightsquigarrow_L}b$ or $a{\rightsquigarrow_L}\bullet{\neq}b{\rightsquigarrow_L}\bullet$. We consider the case where $\bullet{\rightsquigarrow_L}a{\neq}\bullet{\rightsquigarrow_L}b$ for concreteness, but the argument for $a{\rightsquigarrow_L}\bullet{\neq}b{\rightsquigarrow_L}\bullet$ is analogous.

Without loss of generality, let us assume there exists an activity c such that $c \in \bullet\rightsquigarrow_L a$ and $c \notin \bullet\rightsquigarrow_L b$. This means that $c \rightsquigarrow_L a \wedge a \not\rightsquigarrow_L c$ and $c \not\rightsquigarrow_L b \vee b \rightsquigarrow_L c$. The completeness of ρ' implies that $P'_c \prec' P'_a$ and $P'_c \not\prec' P'_b$. Hence, $P'_a \neq P'_b$.

□

Theorem 3 (Validity Guarantee of Maximal Order). *Let $L \in \mathcal{M}(\Sigma^*)$ be an event log. If a valid complete partial order cut of L exists, then a maximal partial order cut of L exists and is guaranteed to be valid.*

Proof. Assume a complete partial order cut $\rho' = (P', \prec')$ of L exists. From Theorem 2, we know that a maximal partial order cut $\rho = (P, \prec)$ of L exists, and we know that ρ is unique by Theorem 1. We prove that ρ adheres to the four conditions of validity (Def. 3) if ρ' is valid.

1) The first condition is satisfied by the definition of completeness.
2) We prove the second condition by contradiction, i.e., assume ρ' is valid and ρ violates the second condition of validity. Since the first condition is satisfied and a partial order is asymmetric, violating the second condition means that there exist $a, b \in \Sigma_L$ with $P_a \neq P_b$, $P_a \not\prec P_b \wedge P_b \not\prec P_a$, and $a \not\rightsquigarrow_L b$.
   We know by Lemma 1 that the subsets of P' are subsets of the subsets of P. Therefore, a and b must be in separate subsets in P' ($P'_a \neq P'_b$). We distinguish four cases depending on the relationship between P'_a and P'_a with respect to $\prec'$:
   - The case $(P'_a \prec' P'_b \wedge P'_b \prec' P'_a)$ is not possible due to the asymmetry of $\prec'$.
   - If $P'_a \not\prec' P'_b$ and $P'_a \not\prec' P'_b$, then ρ' is not valid as $P'_a \neq P'_b$ and $a \not\rightsquigarrow_L b$.
   - If $P'_a \prec' P'_b \wedge P'_b \not\prec' P'_a$, then ρ' is not valid as $a \not\rightsquigarrow_L b$.
   - If $P'_a \not\prec' P'_b \wedge P'_b \prec' P'_a$, then it follows by the first condition of validity that $b \rightsquigarrow_L a$. Since $a \not\rightsquigarrow_L b$ and $b \rightsquigarrow_L a$, the completeness of the maximal order cut ρ implies that $P_b \prec P_a$.

   All cases contradict our assumption.
3) We prove the third condition by contradiction, i.e., assume ρ' is valid and ρ violates the third condition of validity. Then there exists $b \in \Sigma_L$ such that $\nexists\, P_x \in P\Big(P_x \prec P_b\Big)$ and $P_b \cap L_{\triangleright} = \emptyset$.
   By Lemma 1, we know that $P'_b \subseteq P_b$. Since ρ' is valid, we know that if no subset precedes P'_b, then $P'_b \cap L_{\triangleright} \neq \emptyset$. However, since $P'_b \subseteq P_b$ and $P_b \cap L_{\triangleright} = \emptyset$, we conclude that there must exist $a \in \Sigma_L$ such that $P'_a \prec' P'_b$.
   By the definition of completeness, we know that $a \rightsquigarrow_L b \wedge b \not\rightsquigarrow_L a$. Since $a \in P_a$, we conclude by the definition of completeness that $P_a \prec P_b$, i.e., $\exists\, P_x \in P\Big(P_x \prec P_b\Big)$.
4) The proof of the fourth condition is analogous to the third condition.

□

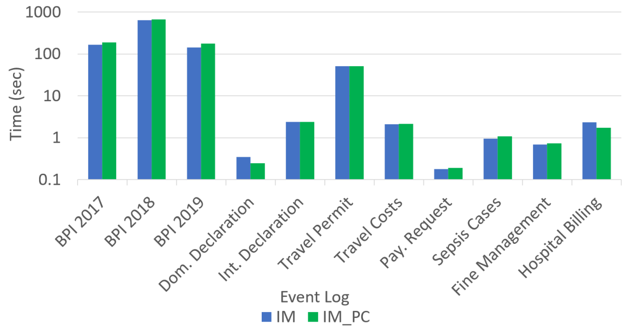

Fig. 6: Time performance results for the unfiltered logs. Both IM and IM$_{\text{PC}}$ are able to scale well on the unfiltered logs.

V. Evaluation

We implemented our discovery approach in PM4Py [6], and we evaluate it using real-life event logs. We compare our approach (IM$_{\text{PC}}$) with the base inductive miner that guarantees prefect fitness (IM) and the brute-force approach for the discovery of POWL models from [15] (IM$_{\text{P}}$).

We transform the discovered models into WF-nets, and we assess their quality using three conformance-checking metrics: fitness [5], precision [24], and simplicity [7].

We use real-life event logs for the evaluation ([19], [18], [20], [8], [11], [9], [10]). Since the approach for the discovery of POWL models from [15] (IM$_{\text{P}}$) is not able to scale well on event logs with a large number of activities, we filter the event logs to only keep the most frequent activities using two values for this filter: 8 and (at most) 12 activities. We additionally apply IM and IM$_{\text{PC}}$ to the unfiltered logs.

Results

The time performance results for the unfiltered logs are shown in Fig. 6, and the results for the filtered logs are shown in Fig. 7. Unlike IM$_{\text{P}}$, our approach is able to scale well on event logs with large numbers of activities. In general, the time values observed for IM$_{\text{PC}}$ are close to the time values observed for IM, while IM$_{\text{P}}$ takes excessively long for some event logs. For example, IM$_{\text{PC}}$ required 0.56 seconds to discover a model for the Travel Permit event log with 12 activities, compared to 280 seconds required by IM$_{\text{P}}$ and 0.45 seconds required by IM. For the unfiltered BPI Challenge 2018, which contains 41 activities, IM$_{\text{PC}}$ required 667 seconds to discover a model compared to 646 seconds required by IM.

The results obtained for the conformance checking metrics are reported in Tab. I. We report the precision and simplicity scores we obtained, and we omit the fitness values as all models achieved a fitness of 1. These results were expected since all three discovery approaches guarantee perfect fitness.

For all event logs, the simplicity of the models discovered by IM$_{\text{PC}}$ is higher or equal to the simplicity of the models discovered by both IM and IM$_{\text{P}}$. This increase in simplicity was expected due to the notion of maximality we defined; a maximal partial order cut is simpler than any other complete partial order cut as it clusters concurrent nodes in the same

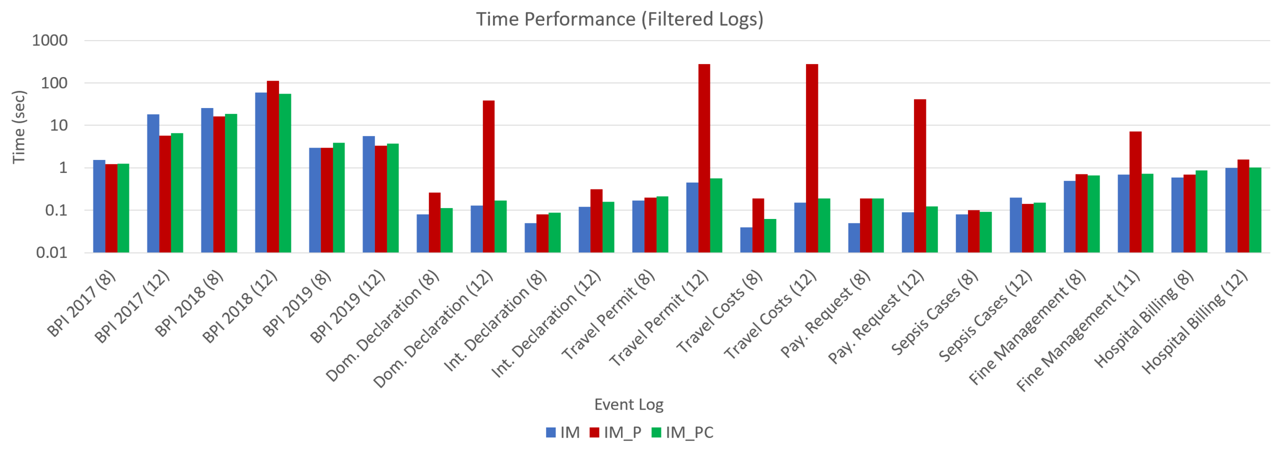

Fig. 7: Time performance results for the filtered logs. The time values observed for IM_{PC} are close to the time values observed for IM, while for IM_{P} we observe some extreme values.

TABLE I: Precision and simplicity results. We highlighted the highest precision and simplicity values observed for each log in green. In general, IM_{PC} led to the models of the highest simplicity and precision.

Event log	#Act.	Precision			Simplicity		
		IM	IM_{P}	IM_{PC}	IM	IM_{P}	IM_{PC}
BPI 2017	8	0.37	0.68	0.68	0.67	0.74	0.74
BPI 2017	12	0.23	0.34	0.34	0.65	0.68	0.69
BPI 2018	8	0.35	0.32	0.32	0.65	0.63	0.65
BPI 2018	12	0.2	0.21	0.23	0.61	0.63	0.64
BPI 2019	8	0.62	0.78	0.78	0.64	0.67	0.67
BPI 2019	12	0.55	0.7	0.7	0.64	0.64	0.65
Dom. Declaration	8	0.4	0.4	0.4	0.65	0.66	0.66
Dom. Declaration	12	0.5	0.54	0.54	0.61	0.67	0.67
Int. Declaration	8	0.5	0.53	0.53	0.67	0.71	0.71
Int. Declaration	12	0.47	0.51	0.51	0.65	0.69	0.69
Travel Permit	8	0.51	0.51	0.51	0.67	0.67	0.67
Travel Permit	12	0.33	0.35	0.36	0.65	0.67	0.67
Travel Costs	8	0.43	0.39	0.43	0.63	0.68	0.7
Travel Costs	12	0.23	0.35	0.23	0.58	0.68	0.69
Pay. Request	8	0.75	0.75	0.75	0.63	0.68	0.68
Pay. Request	12	0.49	0.49	0.49	0.6	0.67	0.67
Sepsis Cases	8	0.5	0.5	0.5	0.64	0.65	0.65
Sepsis Cases	12	0.34	0.35	0.35	0.64	0.65	0.65
Fine Management	8	0.76	0.76	0.76	0.66	0.67	0.67
Hospital Billing	8	0.78	0.78	0.78	0.67	0.67	0.67
Hospital Billing	12	0.6	0.6	0.6	0.65	0.66	0.66

subset, minimizing the number of connections in the graph representation of the partial order.

We observe that precision varies among the different event logs. On the one hand, IM_{PC} led to higher precision than IM_{P} for three event logs. Such results were expected due to the notion of maximality of partial order cuts, where concurrent activities are clustered together, allowing for the discovery of more structures in the next iterations. For example, Fig. 8 shows a sub-model of the POWL model discovered by IM_{P} for the BPI Challenge 2018 event log with 12 activities, and Fig. 9 shows the sub-model discovered by IM_{PC} for the projection of the log on the same activities. IM_{PC} merged concurrent subsets together, leading to the discovery of the sequential dependency "decide" $\rightarrow$ "begin payment" $\rightarrow$ "finish payment".

On the other hand, we observe one case (Travel Costs with 12 activities) where IM_{P} performed better than IM_{PC} in terms of precision. Such results were expected as we restricted the search space to mine for complete partial orders. For this log, IM_{PC} failed to discover a complete partial order, while IM_{P} returned an incomplete partial order cut consisting of two unconnected subsets (in the form $\rho = (\{\{a_1\}, \{a_2, a_3, ..., a_12\}\}, \emptyset)$. Both IM and M_{PC} invoked the fall-through, which returned a loop cut (i.e., tau-loop fall-through [16]).

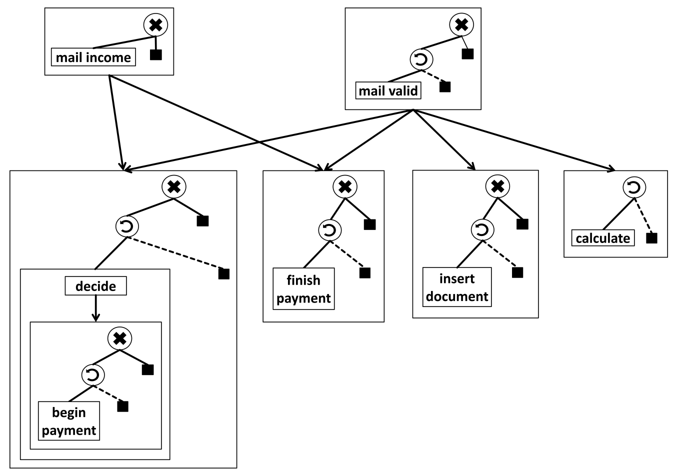

Fig. 8: A POWL model discovered by IM_{P} for the projection of the BPI Challenge 2018 event log on seven activities.

By comparing the results of IM_{PC} with IM, we observe that IM_{PC} led to higher or equal precision values for all event logs except BPI Challenge 2018 with 8 activities. For this case, invoking the fall-through led to better results than detecting a partial order; in order to continue the recursion, the fall-through returned a concurrency cut between an activity that occurs in every trace at most once and the rest of the activities.

VI. Conclusion

In his paper, we proposed a scalable approach for the discovery of Partially Ordered Workflow Language (POWL)

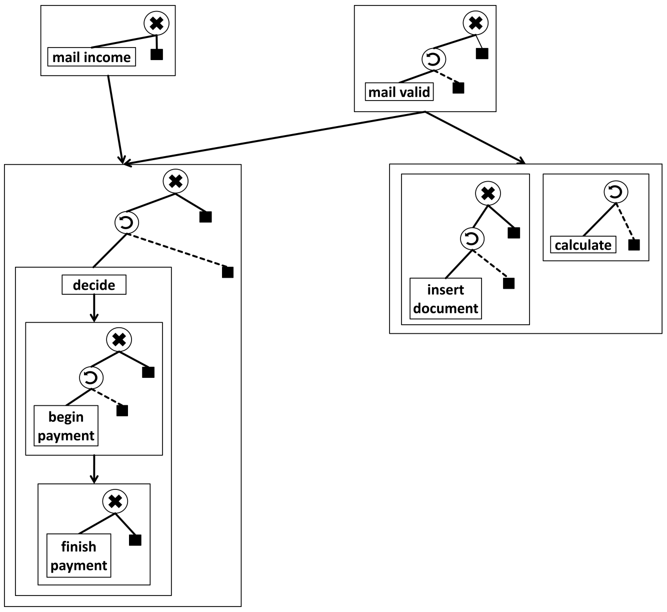

Fig. 9: A POWL model discovered by IM_{PC} for the same log used in Fig. 8. Concurrent activities are subsets together, leading to the discovery of more structures inside these subsets.

models. Prior to this work, there was no existing discovery technique for POWL models that is scalable enough to handle large real-life data sets. Beyond scalability, our discovery approach provides formal guarantees on the uniqueness, existence, and quality of the discovered partial orders. We believe that our approach contributes to the robustness and dependability of POWL models in practice. We evaluated our approach using real-life event logs, and the evaluation showed that our approach is able to scale well on large data sets while ensuring high quality for the discovered models.

We propose multiple ideas for future work. Firstly, we recognize the need for filtering mechanisms. In most real-life event logs, a large fraction of behavior is covered by a small fraction of trace variants (i.e., unique activity sequences). Filtering to retain only the most frequent trace variants could significantly improve the precision of the discovered models. Secondly, enhancing the visualization of POWL models is a crucial aspect of future work. Thirdly, our approach extends the base inductive miner to mine for partial orders after mining for process tree cuts. One possible extension is to adapt the approach to directly mine for POWL models without trying to detect sequence and concurrency process tree cuts first. Moreover, we can develop a discovery approach for POWL models that exploits life-cycle information in event logs where each event has a duration (i.e., each event has a start timestamp and an end timestamp). Finally, the idea of combining different modeling notations to create new types of process models is not restricted to POWL models. This idea can be applied to combine other types of process models.

References

[1] van der Aalst, W.M.P.: The application of petri nets to workflow management. J. Circuits Syst. Comput. **8**(1), 21–66 (1998)

[2] van der Aalst, W.M.P., De Masellis, R., Di Francescomarino, C., Ghidini, C., Kourani, H.: Discovering hybrid process models with bounds on time and complexity: When to be formal and when not? Information Systems **116**, 102214 (2023)

[3] Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019)

[4] Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of petri nets from infinite partial languages. In: Billington, J., Duan, Z., Koutny, M. (eds.) ACSD 2008. pp. 170–179. IEEE (2008)

[5] Berti, A., van der Aalst, W.M.P.: Reviving token-based replay: Increasing speed while improving diagnostics. In: van der Aalst, W.M.P., Bergenthum, R., Carmona, J. (eds.) Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, Satellite event of Petri Nets 2019 and ACSD 2019. CEUR Workshop Proceedings, vol. 2371, pp. 87–103. CEUR-WS.org (2019)

[6] Berti, A., van Zelst, S., Schuster, D.: PM4Py: A process mining library for python. Software Impacts **17**, 100556 (2023)

[7] Blum, F.R.: Metrics in process discovery. Tech. Rep. TR/DCC-2015-6, Computer Science Department, Universidad de Chile, Chile (2015)

[8] van Dongen, B.: BPI Challenge 2017 (2017)

[9] van Dongen, B.: BPI Challenge 2019 (2019)

[10] van Dongen, B.: BPI Challenge 2020 (2020)

[11] van Dongen, B., Borchert, F.: BPI Challenge 2018 (2018)

[12] van Dongen, B.F., Desel, J., van der Aalst, W.M.P.: Aggregating causal runs into workflow nets. Trans. Petri Nets Other Model. Concurr. **6**, 334–363 (2012)

[13] Dumas, M., García-Bañuelos, L.: Process mining reloaded: Event structures as a unified representation of process models and event logs. In: Application and Theory of Petri Nets and Concurrency - 36th International Conference, Proceedings. LNCS, vol. 9115, pp. 33–48. Springer (2015)

[14] Golani, M., Pinter, S.S.: Generating a process model from a process audit log. In: Business Process Management, International Conference, Proceedings. LNCS, vol. 2678, pp. 136–151. Springer (2003)

[15] Kourani, H., van Zelst, S.J.: POWL: partially ordered workflow language. In: Francescomarino, C.D., Burattin, A., Janiesch, C., Sadiq, S. (eds.) BPM 2023. LNCS, vol. 14159, pp. 92–108. Springer (2023)

[16] Leemans, S.J.J.: Robust Process Mining with Guarantees - Process Discovery, Conformance Checking and Enhancement, LNBIP, vol. 440. Springer (2022)

[17] Leemans, S.J., van Zelst, S.J., Lu, X.: Partial-order-based process mining: a survey and outlook. Knowl Inf Syst (2022)

[18] de Leoni, M.M., Mannhardt, F.: Road Traffic Fine Management Process (2015)

[19] Mannhardt, F.: Sepsis Cases - Event Log (2016)

[20] Mannhardt, F.: Hospital Billing - Event Log (2017)

[21] Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Min. Knowl. Discov. **1**(3), 259–289 (1997)

[22] Mokhov, A., Carmona, J.: Event log visualisation with conditional partial order graphs: from control flow to data. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, Satellite event of Petri Nets 2015 and ACSD 2015. CEUR Workshop Proceedings, vol. 1371, pp. 16–30. CEUR-WS.org (2015)

[23] Mokhov, A., Yakovlev, A.: Conditional partial order graphs: Model, synthesis, and application. IEEE Trans. Computers **59**(11), 1480–1493 (2010)

[24] Munoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. Proceedings. LNCS, vol. 6336, pp. 211–226. Springer (2010)

[25] Slaats, T., Schunselaar, D.M.M., Maggi, F.M., Reijers, H.A.: The semantics of hybrid process models. In: On the Move to Meaningful Internet Systems: OTM 2016 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2016, Proceedings. LNCS, vol. 10033, pp. 531–551 (2016)

[26] van der Werf, J.M.E.M., Polyvyanyy, A., van Wensveen, B.R., Brinkhuis, M.J.S., Reijers, H.A.: All that glitters is not gold: Four maturity stages of process discovery algorithms. Inf. Syst. **114**, 102155 (2023)