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 [15] is a block-structured process modeling notation that uses control-flow operators (sequence →, choice ×, concurrency +, and loop ⊙) 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.

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 × and ⊙ 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 [20], 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 [25]). 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 confor-
mity 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 notation 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 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 [1] 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 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

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

A multi-set generalizes the notion of a set and allows for multiple occurrences of the same element. We define a multi-set \( M : \mathbb{N} \to \mathbb{N} \) as a function from \( \mathbb{N} \) to \( \mathbb{N} \), where \( M(i) \) is the multiplicity of the \( i \)-th element in the multi-set. The set \( \mathbb{N} \times \mathbb{N} \) generalizes the notion of a set and allows for multiple occurrences of the same element. We define a multi-set \( M : \mathbb{N} \to \mathbb{N} \) as a function from \( \mathbb{N} \) to \( \mathbb{N} \), where \( M(i) \) is the multiplicity of the \( i \)-th element in the multi-set. The set \( \mathbb{N} \times \mathbb{N} \) generalizes the notion of a set and allows for multiple occurrences of the same element. We define a multi-set \( M : \mathbb{N} \to \mathbb{N} \) as a function from \( \mathbb{N} \) to \( \mathbb{N} \), where \( M(i) \) is the multiplicity of the \( i \)-th element in the multi-set. The set \( \mathbb{N} \times \mathbb{N} \) generalizes the notion of a set and allows for multiple occurrences of the same element. We define a multi-set \( M : \mathbb{N} \to \mathbb{N} \) as a function from \( \mathbb{N} \) to \( \mathbb{N} \), where \( M(i) \) is the multiplicity of the \( i \)-th element in the multi-set.

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

B. Event Log

We use \( \Sigma \) 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. \( \Sigma_L = \{a \in \Sigma \mid \exists \sigma \in L, 1 \leq i < |\sigma| (\sigma(i) = a)\} \) denotes the set of activities that occur in \( L \). We use \( L_o = \{a \in \Sigma_L \mid \exists \sigma \in L (\sigma(1) = a)\} \) to denote the set of start activities and \( L_e = \{a \in \Sigma_L \mid \exists \sigma \in L (|\sigma| = i)\} \) to denote the set of end activities. The Directly-Follows Graph (DFG) is a 2-ary relation \( \Rightarrow \subseteq \Sigma_L \times \Sigma_L \) that captures direct successions between activities: \( a \Rightarrow b \) if \( \exists \sigma \in L, 1 \leq i < |\sigma| (\sigma(i) = a \land \sigma(i+1) = b) \). The Eventually-Follows Graph (EFG) \( \Rightarrow^* \subseteq \Sigma_L \times \Sigma_L \) captures direct and indirect successions between activities: \( a \Rightarrow^* b \) if \( \exists \sigma \in L, 1 \leq i < j \leq |\sigma| (\sigma(i) = a \land \sigma(j) = b) \). For an activity \( a \), the pre-set of \( a \) is defined as \( \bullet \Rightarrow a = \{b \mid b \Rightarrow a \land a \neq b\} \), while the post-set of \( a \) is defined as \( a \Rightarrow \bullet = \{b \mid a \Rightarrow L b \land b \neq L a\} \).

For instance, \( L_1 = \{(a, b, c)^3, (a, b, d)^2\} \) is an event log consisting of five traces with \( \Sigma_L = \{a, b, c, d\} \).

\[\text{Irreflexivity and transitivity imply asymmetry (i.e., } z_1 \not\Rightarrow z_2 \Rightarrow z_2 \not\Rightarrow z_1).\]
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 $\odot$ 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 $(\{A_1, \ldots, A_n\}, \preceq)$ over a partitioning of the activities into $n \geq 2$ subsets $A_1, \ldots, A_n$.

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 $L$. $\rho$...
A partial order cut is valid if the following conditions hold for all \( A_i, A_j \in P \):\( A_i \neq A_j \):

1. \( A_i \prec A_j \) if and only if \( A_i \prec A_j \) in the eventually-follows graph.

2. \( A_i \prec A_j \) and \( A_j \prec A_i \) are both valid.

3. If \( \exists k \in P \) such that \( A_k \prec A_i \), then \( A_i \cap L \neq \emptyset \).

4. If \( \not\exists k \in P \) such that \( A_k \prec A_i \), then \( A_i \cap L \neq \emptyset \).

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 \( + \) 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 \( \rho_2 \) to be better than \( \rho_1 \) as \( \rho_1 \) 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 \( L \). \( \rho \) is complete if the following conditions hold for all \( A_i, A_j \in P \); \( a_i \in A_i \); \( a_j \in A_j \):

- \( A_i \prec A_j \) if and only if \( a_i \prec a_j \) in the eventually-follows graph.

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., \( a \prec_L b \land b \not\prec_L a \) and \( \{a, b\} \notin \{\{a\}, \{b\}\} \). 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.
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 $\bullet \sim_L a_i = \bullet \sim_L a_j$ and $a_i \sim_L \bullet = a_j \sim_L \bullet$. Since activities within the same subset of the partitioning share the same predecessors and successors, the completeness condition (Def. 5) uniquely defines a binary relation $\prec$ over the partitioning: $A_i \prec A_j$ iff $\forall a_i \in A_i, a_j \in A_j \left( a_i \sim_L a_j \wedge a_j \not\sim_L a_i \right)$.

The uniquely defined binary relation $\prec$ is guaranteed to be irreflexive since it is impossible to fulfill $a \sim_L a \wedge a \not\sim_L a$ 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.

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 $\rho$ 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 $L$, 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 $a \sim_L b \wedge b \not\sim_L a \wedge b \sim_L c \wedge c \not\sim_L b$, and $a \not\sim_L c \vee c \not\sim_L a$.

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

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 $\rho$.

**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 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 $\bullet \sim_L a \neq \bullet \sim_L b$ or $a \sim_L \bullet \neq b \sim_L \bullet$. We consider the case where $\bullet \sim_L a \neq \bullet \sim_L b$ for concreteness, but the argument for $a \sim_L \bullet \neq b \sim_L \bullet$ is analogous.
Without loss of generality, let us assume there exists an activity \( c \) such that \( c \in \cdot \rightarrow_L a \) and \( c \notin \cdot \rightarrow_L b \). This means that \( c \cdot \rightarrow_L a \land a \cdot \rightarrow_L c \) and \( c \cdot \rightarrow_L b \lor b \cdot \rightarrow_L c \). The completeness of \( \rho' \) implies that \( P'_c \prec P'_a \) and \( P'_c \not\succ P'_b \). Hence, \( P'_a \neq P'_b \).

**Theorem 3 (Validity Guarantee of Maximal Order).** Let \( L \in M(\Sigma^*) \) 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 \( \rho \) is unique by **Theorem 1**. We prove that \( \rho \) adheres to the four conditions of validity (Def. 3) if \( \rho' \) is valid.

1. The first condition is satisfied by the definition of completeness.
2. We prove the second condition by contradiction, i.e., assume \( \rho' \) is valid and \( \rho \) 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 \( P_a \neq P_b \) and \( P_a \neq P_b \land P_a \neq P_b \). 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 \( P'_b \) with respect to \( \neq' \):
   - The case \( P'_a \neq P'_b \) is not possible due to the asymmetry of \( \neq' \).
   - If \( P'_a \neq P'_b \) and \( P'_a \neq P'_b \), then \( \rho' \) is not valid as \( P'_a \neq P'_b \land a \neq' L b \).
   - If \( P'_a \neq P'_b \) and \( P'_a \neq P'_b \), then \( \rho' \) is not valid as \( a \neq' L b \).
   - If \( P'_a \neq P'_b \) and \( P'_a \neq P'_b \), then it follows by the first condition of validity that \( b \cdot \rightarrow_L a \). Since \( a \neq' L b \) and \( b \cdot \rightarrow_L a \), the completeness of the maximal order cut \( \rho \) implies that \( P_b \sim P_a \).

All cases contradict our assumption.

3. We prove the third condition by contradiction, i.e., assume \( \rho' \) is valid and \( \rho \) violates the third condition of validity. Then there exists \( b \in \Sigma_L \) such that \( \nexists P_b \in P \left( P \ast \prec P_b \right) \) and \( P_b \cap L_o = \emptyset \).

By **Lemma 1**, we know that \( P'_b \subseteq P_b \). Since \( \rho' \) is valid, we know that if no subset precedes \( P'_b \), then \( P'_b \cap L_o = \emptyset \). However, since \( P'_b \subseteq P_b \) and \( P_b \cap L_o = \emptyset \), we conclude that there must exist \( a \in \Sigma_L \) such that \( P_a \sim P'_b \). By the definition of completeness, we know that \( a \cdot \rightarrow_L b \land b \cdot \rightarrow_L a \). Since \( a \in P_a \), we conclude by the definition of completeness that \( P_a \sim P_b \), i.e., \( \exists P_b \in P \left( P_b \ast \prec P_b \right) \).

4. The proof of the fourth condition is analogous to the third condition.

\[ \square \]
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.

<table>
<thead>
<tr>
<th>Event log</th>
<th>#Act.</th>
<th>Precision</th>
<th>Simplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>IM \ IM(<em>P) \ IM(</em>{PC})</td>
<td>IM \ IM(<em>P) \ IM(</em>{PC})</td>
</tr>
<tr>
<td>BPI 2017</td>
<td>8</td>
<td>0.37 0.68 0.68</td>
<td>0.67 0.74 0.74</td>
</tr>
<tr>
<td>BPI 2017</td>
<td>12</td>
<td>0.23 0.34 0.34</td>
<td>0.65 0.68 0.69</td>
</tr>
<tr>
<td>BPI 2018</td>
<td>8</td>
<td>0.33 0.32 0.32</td>
<td>0.65 0.63 0.65</td>
</tr>
<tr>
<td>BPI 2018</td>
<td>12</td>
<td>0.21 0.21 0.21</td>
<td>0.61 0.63 0.64</td>
</tr>
<tr>
<td>BPI 2019</td>
<td>8</td>
<td>0.62 0.78 0.78</td>
<td>0.64 0.67 0.67</td>
</tr>
<tr>
<td>BPI 2019</td>
<td>12</td>
<td>0.55 0.7 0.7</td>
<td>0.64 0.64 0.65</td>
</tr>
<tr>
<td>Dom. Declaration</td>
<td>8</td>
<td>0.4 0.4 0.4</td>
<td>0.65 0.66 0.66</td>
</tr>
<tr>
<td>Dom. Declaration</td>
<td>12</td>
<td>0.5 0.54 0.54</td>
<td>0.61 0.67 0.67</td>
</tr>
<tr>
<td>Int. Declaration</td>
<td>8</td>
<td>0.13 0.53 0.33</td>
<td>0.67 0.71 0.71</td>
</tr>
<tr>
<td>Int. Declaration</td>
<td>12</td>
<td>0.47 0.51 0.51</td>
<td>0.65 0.69 0.69</td>
</tr>
<tr>
<td>Travel Permit</td>
<td>8</td>
<td>0.51 0.51 0.51</td>
<td>0.67 0.67 0.67</td>
</tr>
<tr>
<td>Travel Permit</td>
<td>12</td>
<td>0.33 0.35 0.36</td>
<td>0.65 0.67 0.67</td>
</tr>
<tr>
<td>Travel Costs</td>
<td>8</td>
<td>0.43 0.39 0.43</td>
<td>0.63 0.68 0.68</td>
</tr>
<tr>
<td>Travel Costs</td>
<td>12</td>
<td>0.23 0.35 0.23</td>
<td>0.58 0.68 0.69</td>
</tr>
<tr>
<td>Pay. Request</td>
<td>8</td>
<td>0.75 0.75 0.75</td>
<td>0.63 0.68 0.68</td>
</tr>
<tr>
<td>Pay. Request</td>
<td>12</td>
<td>0.49 0.49 0.49</td>
<td>0.6 0.67 0.67</td>
</tr>
<tr>
<td>Sepsis Cases</td>
<td>8</td>
<td>0.3 0.3 0.5</td>
<td>0.64 0.65 0.65</td>
</tr>
<tr>
<td>Sepsis Cases</td>
<td>12</td>
<td>0.34 0.35 0.35</td>
<td>0.64 0.65 0.65</td>
</tr>
<tr>
<td>Fine Management</td>
<td>8</td>
<td>0.76 0.76 0.76</td>
<td>0.66 0.67 0.67</td>
</tr>
<tr>
<td>Hospital Billing</td>
<td>8</td>
<td>0.78 0.78 0.78</td>
<td>0.67 0.67 0.67</td>
</tr>
<tr>
<td>Hospital Billing</td>
<td>12</td>
<td>0.6 0.6 0.6</td>
<td>0.65 0.66 0.66</td>
</tr>
</tbody>
</table>

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 \(\rho = (\{a_1\}, \{a_2, a_3, \ldots, a_2\}, \emptyset)\)). Both IM and M\(_{PC}\) invoked the fall-through, which returned a loop cut (i.e., tau-loop fall-through [16]).

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)
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.

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

REFERENCES