Mining for Long-Term Dependencies in Causal Graphs

Humam Kourani[1], Chiara Di Francescomarino[2], Chiara Ghidini[2], Wil van der Aalst[3], and Sebastiaan van Zelst[1]

[1] Fraunhofer FIT - Data Science and Artificial Intelligence, Sankt Augustin, Germany
{humam.kourani,sebastiaan.van.zelst}@fit.fraunhofer.de

[2] Fondazione Bruno Kessler - Process and Data Intelligence, Trento, Italy
{dfmchiara,ghidini}@fbk.eu

[3] RWTH Aachen University - Process and Data Science, Aachen, Germany
wvdaalst@pads.rwth-aachen.de

Abstract. Process discovery is one of the most challenging tasks in process mining. Based on event data, a process discovery approach generates a process model that captures the behavior recorded in the data. The hybrid miner is a two-step process discovery approach that creates a balance between the advantages of formal modeling and the necessity of remaining informal for vague structures. In the first discovery step, an informal causal graph is constructed based on direct succession dependencies between activities. In the second discovery step, the hybrid miner tries to convert the discovered dependencies into formal constraints. For vague structures where formal constraints cannot be justified, dependencies are depicted informally. In this paper, we reduce the representational bias of the hybrid miner by exploiting causal graph metrics to mine for long-term dependencies. Our evaluation shows that the proposed approach leads to the discovery of more precise models.

Key words: causal graphs, long-term dependencies, hybrid miner

1 Introduction

Process mining is a family of techniques that can be applied to analyze and monitor systems based on the events they produce. *Process discovery* is one of the main branches of process mining. Process discovery techniques analyze event data, aiming at discovering a process model capturing the behavior recorded in the data; the resulting process model illustrates how process activities are related to each other [6]. Most existing process discovery techniques produce *formal models* that have executable semantics and are able to classify traces into *fitting* and *non-fitting*. Alternatively, most commercial process mining tools use *informal models* that illustrate causal dependencies between activities without providing executable semantics. Although formal models provide more powerful insights, commercial tools favor representing processes using informal models due to multiple reasons. First of all, attempting to formally model complex structures results in complex models that cannot be easily interpreted by users. Moreover, for most real-life processes there is no clear correct classification of traces into fitting and non-fitting due to noise and infrequent behavior. Trying to precisely model all behavior seen in an event log can lead to overfitting process models that are not able to generalize well on unseen data. Finally, the discovery of formal models is very time-consuming compared to the discovery of informal models. Commercial process mining tools need to handle huge logs and interactively generate and update process models based on them.

The *hybrid miner* [7] combines the best of formal and informal modeling notations by discovering *hybrid Petri nets*. A hybrid Petri net shows some causal dependencies in an informal way similar

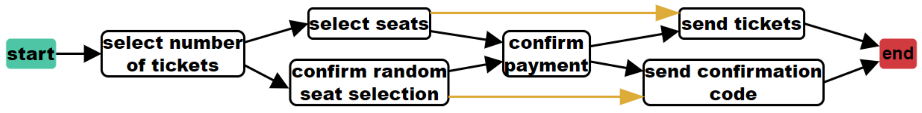


Fig. 1: Causal graph extended to include long-term dependencies.

to the models produced by commercial tools, and at the same time, it contains *places* that provide formal semantics for other parts of the process. The hybrid miner is a two-step discovery approach. In the first discovery step, causal metrics are computed based on direct succession dependencies between activities, and a causal graph is generated based on these metrics. A *causal graph* consists of nodes representing activities and two types of directed edges connecting nodes. *Certain edges* represent strong causal dependencies and *uncertain edges* represent weak dependencies. In the second discovery step, the hybrid miner converts the discovered certain edges into formal places if there is enough evidence in the data justifying adding formal constraints. For vague structures where formal constraints cannot be justified, causal relations are depicted in the final hybrid Petri net as informal edges. Since uncertain edges represent weak dependencies, they are not used for building places; they are added to the final Petri net as informal edges as well.

One of the main limitations of the hybrid miner is that it is not able to detect long-term dependencies. The hybrid miner constructs the causal graph based on direct succession dependencies between activities, preventing the discovery of simple structures with long-term dependencies. For instance, let us consider a simple process of booking concert tickets. After selecting the number of tickets, customers either select their seats (against additional fees) or confirm a random seat allocation. Afterward, customers confirm their order by paying. Finally, based on the earlier decision, customers either directly receive their tickets with assigned seat numbers via email or receive a confirmation code that they should use on the day of the concert to get their tickets. Figure 1 shows a causal graph discovered in the first step of the hybrid miner to model this process extended with two additional edges modeling long-term dependencies (visualized through yellow arcs). The causal graph shows dependencies between activities in an informal manner. The additional long-term dependency edges cannot be discovered by the hybrid miner because certain edges are constructed based on direct succession metrics (*select seats* is never directly followed by *send tickets* and *confirm random seat selection* is never directly followed by *send confirmation code*).

Certain edges are transformed into formal constraints in the second discovery step of the hybrid miner as shown in Figure 2. The edges (*confirm payment* $\rightarrow$ *send tickets*) and (*confirm payment* $\rightarrow$ *send confirmation code*) are transformed into the place (visualized through a circle) connecting *confirm payment* with *send tickets* and *send confirmation code*. This place formally models a choice between sending tickets and confirmation codes after the payment. Since this choice depends on the earlier decision, two additional places are needed to capture this non-free-choice: the place connecting *select seats* with *send tickets* and the place connecting *confirm random seat selection* with *send confirmation code*. This behavior cannot be modeled precisely by the hybrid miner because these two places are generated based on the additional yellow edges we added to model long-term dependencies. Without these places, the model would allow, for instance, for behavior where customers select their seats and then receive a confirmation code for a random seat assignment.

In this paper, we propose an approach for extending the first discovery step of the hybrid miner to detect long-term dependencies. The proposed approach keeps using direct succession metrics for creating the initial causal graph, and it mines for long-term dependencies as a post-processing

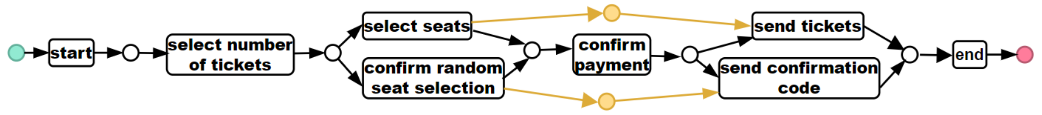


Fig. 2: (Hybrid) Petri net discovered based on Figure 1.

step. We define additional metrics based on eventually-follow relations between activities. We use these metrics to detect and filter long-term dependencies, and we add certain edges based on them. These certain edges are used in the second step for generating candidate places to formally capture long-term dependencies. This helps reduce the representational bias of the hybrid miner.

The remainder of the paper is structured as follows. We present some preliminaries in Section 2. In Section 3, we define an extended version of the causal graph miner that supports the detection of long-term dependencies. We evaluate our approach in Section 4, and we discuss related work in Section 5. Finally, we provide a short summary of the paper in Section 6.

2 Preliminaries

In this section, we address the preliminaries needed to understand the concepts we present in this paper.

2.1 Event Log

In the field of process mining, data is often represented in the form of *events*. The term event refers to the recording of an activity that happened during the execution of a process instance. Each event contains attributes providing information about the executed activity. The term *trace* refers to a sequence of events that represents the execution of a process instance. An *event log* is a collection of events that record the execution of multiple instances of a particular process. For simplicity, we abstract from these notations and define an event log as a multi-set of activity sequences.

Before providing any formal definitions, we introduce basic notations (based on [7]):
- $\mathcal{B}(X)$ denotes the set of all multi-sets over some set X. For example, $M = [{x_1}^2,\ x_2] \in \mathcal{B}(X)$ is a multi-set over $X = \{x_1,\ x_2,\ x_3\}$ with $|M| = 3$.
- X^* denotes the set of all sequences over some set X.
- For a sequence σ, $\sigma(i)$ denotes the i-th element of the sequence and $|\sigma|$ denotes the length of the sequence. For example, $\langle x_1,\ x_2,\ x_1\rangle \in X^*$ is a sequence over $X = \{x_1,\ x_2,\ x_3\}$ with $\sigma(1) = \sigma(3) = x_1$, $\sigma(2) = x_2$, and $|\sigma| = 3$.

Definition 1 (Event Log [7]). *Let $\mathcal{A}$ be a set of activities. A trace $\sigma \in \mathcal{A}^*$ is a sequence of activities. An event log $L \in \mathcal{B}(\mathcal{A}^*)$ is a multi-set of traces.*

$L_1 = [\langle \textit{select number of tickets}, \textit{select seats}, \textit{confirm payment}, \textit{send tickets}\rangle^{70}, \langle \textit{select number of tickets}, \textit{confirm random seat selection}, \textit{confirm payment}, \textit{send confirmation code}\rangle^{30}]$ is an example of an event log that contains 100 traces and 400 events.

2.2 Causal Graph

A causal graph is a directed graph with nodes representing activities and edges representing causal relations between activities. There are two types of edges: *certain* and *uncertain*. The edge type is based on the strength of the causal relation between the two activities connected by the edge. Certain edges are used to represent *strong* causal relations; uncertain edges are used to represent *weak* relations. Note that different metrics can be used to determine the strength of causal relations.

Definition 2 (Causal Graph [7]). *A* causal graph *is a triple* $G = (\mathcal{A}, R_S, R_W)$ *where* $\mathcal{A}$ *is a set of activities,* $R_S \subseteq \mathcal{A} \times \mathcal{A}$ *is the set of certain edges,* $R_W \subseteq \mathcal{A} \times \mathcal{A}$ *is the set of uncertain edges, and* $R_S \cap R_W = \emptyset$*.*

In all examples in this paper, we abstract from uncertain edges because they represent weak dependencies and they are not used for generating candidate places in the second discovery step; We only visualize certain edges. Figure 1 shows an example causal graph.

2.3 Direct Dependency Metrics

There are many possible approaches for constructing a causal graph based on an event log. The hybrid miner uses a variant of the approach used by the heuristic miner [6, 15]. In order to construct a causal graph, we define a causality metric ($Caus_\alpha$) for evaluating causal relations between activities and adding edges accordingly. This metric is based on direct succession dependencies between activities while taking concurrency and loops into account as well. In Definition 3, we define the causality metric $Caus_\alpha$.

Definition 3 (Direct Dependency Metrics [7]). *Let* $L \in \mathcal{B}(\mathcal{A}^*)$ *be an event log over a set of activities* $\mathcal{A}$ *and let* $\{a, b\} \subseteq \mathcal{A}$*.*

- $\#(a,b,L) = \sum_{\sigma \in L} |\{i \in \{1,..,|\sigma|-1\} \mid \sigma(i) = a \ \wedge \ \sigma(i+1) = b\}|$ *counts the number of times a is directly followed by b in L.*
- $\#(a,*,L) = \sum_{\sigma \in L} |\{i \in \{1,..,|\sigma|-1\} \mid \sigma(i) = a\}|$ *counts the number of times a is directly followed by any activity in L.*
- $\#(*,b,L) = \sum_{\sigma \in L} |\{i \in \{2,..,|\sigma|\} \mid \sigma(i) = b\}|$ *counts the number of times b is directly preceded by any activity in L.*
- $Rel1(a,b,L) = \dfrac{\#(a,b,L) + \#(a,b,L)}{\#(a,*,L) + \#(*,b,L)}$ *evaluates the strength of the causal relation* (a,b) *relative to the split and join behavior of activities a and b.*
- $$Rel2(a,b,L) = \begin{cases} \dfrac{\#(a,b,L) - \#(b,a,L)}{\#(a,b,L) + \#(b,a,L) + 1} & \text{if } \#(a,b,L) - \#(b,a,L) > 0 \\ \dfrac{\#(a,b,L)}{\#(a,b,L) + 1} & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$
  evaluates the strength of the causal relation (a,b) *taking into account concurrency and loops.*
- $Caus_\alpha(a,b,L) = \alpha \cdot Rel1(a,b,L) \ + \ (1-\alpha) \cdot Rel2(a,b,L)$ *is the weighted average of* $Rel1(a,b,L)$ *and* $Rel2(a,b,L)$ *where* $\alpha \in [0,1]$*.*

The metrics *Rel1*, *Rel2*, and $Caus_\alpha$ all produce values between 0 and 1. Low values indicate weak dependencies; high values indicate strong dependencies. The variable $\alpha \in [0,1]$ sets the weight of the relations *Rel1* and *Rel2* in $Caus_\alpha$. Let us consider our example log L_1 and the causal graph shown in Figure 1. Both long-term dependency edges (yellow arcs) achieve a $Caus_\alpha$ score of 0 regardless of the value of α. Therefore, the hybrid miner is not able to discover these edges.

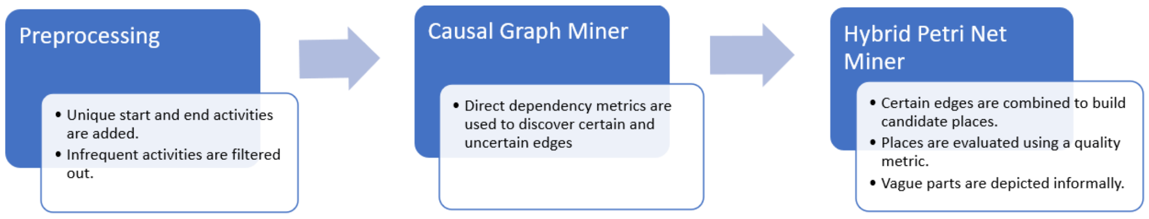

Preprocessing
• Unique start and end activities are added.
• Infrequent activities are filtered out.
Causal Graph Miner
• Direct dependency metrics are used to discover certain and uncertain edges
Hybrid Petri Net Miner
• Certain edges are combined to build candidate places.
• Places are evaluated using a quality metric.
• Vague parts are depicted informally.

Fig. 3: Three steps of the hybrid Miner.

2.4 Hybrid Miner

The hybrid miner is a two-step process discovery approach that constructs a hybrid Petri net based on an input event log. The hybrid miner works according to the three steps shown in Figure 3.

First, the hybrid miner preprocesses the log by adding a unique start activity "start" and a unique end activity "end" to all traces in order to help make the discovered models more understandable for users[1]. Moreover, the user sets a filtering threshold, and infrequent activities are filtered out based on it. In the first discovery step, a causal graph is discovered based on the causality metric $Caus_{\alpha}$ defined in Definition 3. A parameter t_{R_S} is used for setting a threshold for certain edges. Similarly, another parameter t_{R_W} is used for uncertain edges [2]. An additional parameter is used to set the value of the weight α. In Section 3, we will formally define an extended version of the causal graph miner that detects long-term dependencies.

In the second discovery step, the hybrid miner transforms the discovered causal graph into a hybrid Petri net. Candidate places are generated by combining certain edges, and they are evaluated using a quality metric. A parameter t_{eval} is used for setting a threshold for the quality metric. Based on this evaluation, some places are accepted and added to the hybrid Petri net while other places are rejected. At the end of the second discovery step, uncertain edges as well as certain edges that are not covered by any accepted places are added to the hybrid Petri net as informal edges. As the second discovery step is beyond the scope of this paper, we refer to [7] for more details on hybrid Petri nets and the second discovery step of the hybrid miner.

3 Detection of Long-Term Dependencies

In this section, we introduce an extended version of the causal graph miner; we extend the causal graph miner to detect and filter long-term dependencies.

3.1 Quantifying Long-Term Dependencies

The causal graph miner constructs causal edges based on direct succession relations between activities. In order to detect long-term dependencies, we also consider *eventually-follow relations*. In Definition 4, we define *outgoing and incoming dependency metrics* based on both direct succession relations and eventually-follow relations between activities. For a pair of activities (a, b), the *Out-*

[1] In all examples in this paper, we assume that all traces in any event log start with the activity "start" and end with the activity "end" without explicitly mentioning them.

[2] For all examples and experiments in this paper, we use $t_{R_W} = 1$ to deactivate the detection of uncertain edges because these edges are out of the scope of this paper

going Direct Dependency $ODD(a, b, L)$ is an estimation of the probability of executing b directly after executing a. The *Incoming Direct Dependency* $IDD(a, b, L)$ is an estimation of the probability of executing a directly before executing b. The *Outgoing Long-term Dependency* $OLD(a, b, L)$ is an estimation of the probability of eventually executing b after executing a. The *Incoming Long-term Dependency* $ILD(a, b, L)$ is an estimation of the probability of eventually executing a before executing b.

Definition 4 (Outgoing and Incoming Dependency Metrics). *Let* $L \in \mathcal{B}(\mathcal{A}^*)$ *be an event log over a set of activities* $\mathcal{A}$ *and* $\{a, b\} \subseteq \mathcal{A}$*.*

- $ODD(a, b, L) = \dfrac{\#(a, b, L)}{\#(a, *, L)}$ *is the outgoing direct dependency score of* (a, b) *in* L*.*
- $IDD(a, b, L) = \dfrac{\#(a, b, L)}{\#(*, b, L)}$ *is the incoming direct dependency score of* (a, b) *in* L*.*
- $\widehat{\#}(a, L) = \left|\{\sigma \in L \mid \exists_{1 \leq i \leq |\sigma|}\ \sigma(i) = a\}\right|$ *counts the number of traces in* L *where* a *occurs.*
- $\widehat{\#}(a, b, L) = \left|\{\sigma \in L \mid \exists_{1 \leq i \leq j \leq |\sigma|}\ \sigma(i) = a \ \wedge \ \sigma(j) = b\}\right|$ *counts the number of traces in* L *where* a *is eventually followed by* b*.*
- $OLD(a, b, L) = \dfrac{\widehat{\#}(a, b, L)}{\widehat{\#}(a, L)}$ *is the outgoing long-term dependency score of* (a, b) *in* L*.*
- $ILD(a, b, L) = \dfrac{\widehat{\#}(a, b, L)}{\widehat{\#}(b, L)}$ *is the incoming long-term dependency score of* (a, b) *in* L*.*

Note that the dependency metrics ODD, IDD, OLD, and ILD all produce values between 0 and 1. Low values indicate weak dependencies; high values indicate strong dependencies. For a weight $\alpha \in [0, 1]$, we define the *Long-Term Dependency* LD_α as the average of OLD and ILD while taking into account concurrency and loops as well. Using higher values for the weight α means placing more emphasis on the long-term split and join behavior of activities (i.e., it means focusing on the outgoing and incoming long-term dependency scores); using lower values indicates placing more emphasis on the detection of concurrency and loops.

Definition 5 (Long-Term Dependency). *Let* $L \in \mathcal{B}(\mathcal{A}^*)$ *be an event log over a set of activities* $\mathcal{A}$*,* $\{a, b\} \subseteq \mathcal{A}$*, and* $\alpha \in [0, 1]$*. We define the long-term dependency score (*LD_α*) of* (a, b) *in* L *as follows.*

$$LD_\alpha(a, b, L) = \ \alpha \cdot ((OLD(a, b, L) + ILD(a, b, L)) \ / \ 2) + (1 - \alpha) \cdot max\{0, \ \frac{\widehat{\#}(a, b, L) - \widehat{\#}(b, a, L)}{\widehat{\#}(a, b, L) + \widehat{\#}(b, a, L)}\}.$$

For the sake of simplicity, we define an event log $L_2 = [\langle a1,\ c,\ a2\rangle,\ \langle b1,\ c,\ b2\rangle]$ with two straightforward long-term dependencies $(a1,\ a2)$ and $(b1,\ b2)$, and we use this event log in the reminder of the section as our running example. For any $\alpha \in [0, 1]$, both relations $(a1,\ a2)$ and $(b1,\ b2)$ achieve a long-term dependency score (LD_α) of 1.

3.2 Pruning Long-Term Dependencies

Adding all discovered long-term dependencies to the causal graph abundantly increases the number of certain edges and, therefore, abundantly increases the number of candidate places. This has a huge impact on the time performance of the hybrid miner. We can use a threshold for filtering long-term dependencies based on the scores obtained by LD_α; however, this is not sufficient. In our running example, LD_α achieves a score of 1 for the relations $(start,\ c)$, $(c,\ end)$, and $(start,\ end)$.

These relations are clearly not the type of long-term dependencies we are interested in because they are implied by other dependencies. A hybrid Petri net modeling a free-choice between $a1$ and $b1$ after *start* and another free-choice between $a2$ and $b2$ after c fixes the execution order of the activities ($start \rightarrow c \rightarrow end$), and there is no need for additional places for modeling the long-term dependencies. Adding additional certain edges to the causal graph for such long-term dependencies leads to additional time costs without helping improve the quality of the final models. Therefore, we propose further reduction mechanisms to keep only "interesting" long-term dependencies.

We define an extended version of the causal graph miner that mines for long-term dependencies. Long-term dependency relations between activities are evaluated based on the scores obtained by LD_α, and they are then filtered based on the metrics defined in Definition 4. We use a parameter t_{LD} to set a minimum threshold for LD_α. For the weight α, we use the same parameter used to set the weight α for $Caus_\alpha$. We recommend using high values for t_{LD} in order to avoid obtaining "fake" long-term dependencies resulting from loops, noise, or concurrency. We are often interested in long-term dependency relations that achieve high scores for LD_α.

Definition 6 (Extended Causal Graph Miner). *Let $L \in \mathcal{B}(\mathcal{A}^*)$ be an event log over a set of activities $\mathcal{A}$, $\alpha \in [0,1]$, $t_{R_S} \in [0,1]$, $t_{R_W} \in [0,1]$, and $t_{LD} \in [0,1]$. A causal graph $G = (\mathcal{A}, R_S, R_W, R_{LD})$ is discovered for L as follows:*

- $R_{DD} = \{(a,b) \in \mathcal{A} \times \mathcal{A} \mid Caus_\alpha(a,b,L) \geq t_{R_S}\}$ *is the set of direct dependency certain edges.*
- $R_{LD} = \{(a,b) \in \mathcal{A} \times \mathcal{A} \mid a \neq b \;\wedge\; (a,b) \notin R_{DD} \;\wedge\; LD_\alpha(a,b,L) \geq t_{LD} \;\wedge$

$$\forall_{x \in \mathcal{A} \setminus \{a,b\}} OLD(a,b,L) > OLD(a,x,L) \cdot OLD(x,b,L) \;\wedge$$
$$\forall_{x \in \mathcal{A} \setminus \{a,b\}} ILD(a,b,L) > ILD(a,x,L) \cdot ILD(x,b,L) \;\wedge$$
$$OLD(a,b,L) > \frac{\sum_{x \in \mathcal{A} \setminus \{a\}} ODD(a,x,L) \cdot OLD(x,b,L)}{\sum_{x \in \mathcal{A} \setminus \{a\}} ODD(a,x,L)} \;\wedge$$
$$ILD(a,b,L) > \frac{\sum_{x \in \mathcal{A} \setminus \{b\}} ILD(a,x,L) \cdot IDD(x,b,L)}{\sum_{x \in \mathcal{A} \setminus \{b\}} IDD(x,b,L)} \;\}$$

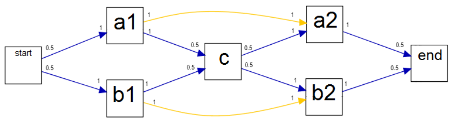
  is the set of long-term dependency edges.
- $R_S = R_{DD} \cup R_{LD}$ *is the set of all certain edges.*
- $R_W = \{(a,b) \in \mathcal{A} \times \mathcal{A} \mid Caus_\alpha(a,b,L) \geq t_{R_W} \;\wedge\; (a,b) \notin R_S\}$ *is the set of uncertain edges.*

3.3 Example Application

In this section, we apply the extended causal graph miner (Definition 6) to our running example in order to investigate the seven conditions used to filter long-term dependencies. Moreover, we apply the approach to another log that covers more advanced structures.

We apply the extended causal graph miner to a the event log $L_2 = [\langle a1,\ c,\ a2\rangle,\ \langle b1,\ c,\ b2\rangle]$ using the parameters $t_{R_S} = t_{LD} = 0.3$ and $\alpha = 0.5$. The discovered causal graph is shown in Figure 4a. The certain edges discovered based on long-term dependencies are visualized using yellow arcs. The causal graph is annotated with outgoing and incoming dependency scores; i.e., long-term dependency edges (R_{LD}) are annotated with *OLD* and *ILD* scores, and other certain edges (R_{DD}) are annotated with *ODD* and *IDD* scores. We use this example to explain the seven conditions that a causal relation must fulfill in order to be accepted as a long-term dependency (i.e., the seven filters used for constructing R_{LD} in Definition 6):

- Self-loops are filtered out. We accept a distance of 0 when computing the eventually-follow score (Definition 4); i.e., each activity is always eventually followed by itself, resulting in a self-loop.
- The second filter ensures avoiding duplicate certain edges. For example, no long-term dependency edge can be discovered for $(a1,\ c)$ because a certain edges is discovered for $(a1,\ c)$ based on direct


(a) Causal graph discovered for L_2.

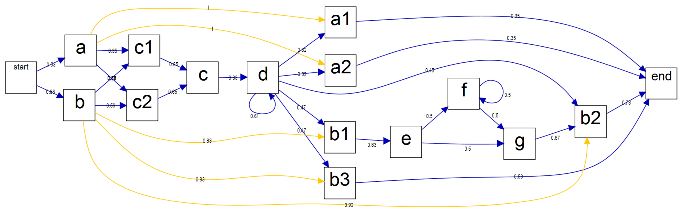

(b) Causal graph discovered for L_3.

Fig. 4: Causal graphs discovered using the extended causal graph miner (Definition 6).

succession dependencies.

- The parameter t_{LD} is used to set a minimum threshold for LD_α.
- The fourth and fifth filters are based on the idea of filtering out a long-term dependency relation if it is covered by other two relations. Outgoing long-term dependency scores are used in the fourth filter; incoming long-term dependency scores are used in the fifth filter. For example, the long-term dependency relation $(start,\ end)$ can be filtered out based on the fourth filter because $OLD(start,\ end,\ L_2) = 1 \not> 1 \cdot 1 = OLD(start,\ c,\ L_2) \cdot OLD(c,\ end,\ L_2)$. All three OLD scores have the value 1 because in all traces *start* is eventually followed by both c and *end*, and c is eventually followed by *end*.
- The last two filters exploit both direct succession dependencies and long-term dependencies. Outgoing dependency scores are used in the sixth filter; incoming dependency scores are used in the seventh filter. The idea is to filter out a long-term dependency if the relation can be covered by the direct neighbors of the source (sixth filter) or by the direct neighbors of the target (seventh filter). For example, the long-term dependency relation $(start,\ c)$ can be filtered out based on the sixth filter. *start* is directly followed by $a1$ with a probability of 0.5 and is directly followed by $b1$ with a probability of 0.5 (i.e., $ODD(start,\ a1,\ L_2) = ODD(start,\ b1,\ L_2) = 0.5$). We do not need to consider further nodes as $a1$ and $b1$ are the only direct neighbors of the source *start* (i.e., other nodes achieve an ODD score of 0). Both $a1$ and $b1$ are always eventually followed by c (i.e., $OLD(a1,\ c,\ L_2) = OLD(b1,\ c,\ L_2) = 1$). Therefore, we obtain: $OLD(start,\ c,\ L_2) = 1 \not> 1 = (0.5 \cdot 1 + 0.5 \cdot 1)/(0.5 + 0.5)$.

A long-term dependency edge was added to the causal graph to represent the causal relation $(a1,\ a2)$ because this relation fulfills all seven conditions:

- $a1 \neq a2$.
- $(a1, a2) \notin R_{DD}$.
- $LD_\alpha(a1, a2, L) = 1 \geq 0.3 = t_{LD}$.
- $OLD(a1, a2, L) = 1 > OLD(a1, x, L) \cdot OLD(x, a2, L)$ for all $x \in \mathcal{A} \setminus \{a1, a2\}$.
- $ILD(a1, a2, L) = 1 > ILD(a1, x, L) \cdot ILD(x, a2, L)$ for all $x \in \mathcal{A} \setminus \{a1, a2\}$.
- $OLD(a1, a2, L) = 1 > 0.5 = \dfrac{1 \cdot 0.5}{1} = \dfrac{ODD(a1, c, L) \cdot OLD(c, a2, L)}{ODD(a1, c, L)}$.
- $ILD(a1, a2, L) = 1 > 0.5 = \dfrac{0.5 \cdot 1}{1} = \dfrac{ILD(a1, c, L) \cdot IDD(c, a2, L)}{IDD(c, a2, L)}$.

In order to test our approach on more complex long-term dependencies, we extend our running example to include loop, choice, and concurrency structures. We define an event log L_3 that consists of the following traces: $\langle a, c1, c2, c, d, d, a1, a2\rangle$, $\langle a, c2, c1, c, d, a2, a1\rangle$, $\langle b, c2, c1, c, d, d, b3\rangle$,

Table 1: Time results of the evaluation of the parameter t_{LD}. We use $t_{0.5}$ (resp. $t_{0.7}$, $t_{0.9}$, and $t_{1.1}$) to denote the results obtained using $t_{LD} = 0.5$ (resp. 0.7, 0.9, and 1.1). We highlight high time values (higher than 5 minutes) in red and moderate time values (between 1 minute and 5 minutes) in yellow.

Log	$\|R_{LD}\|$				**#Places**				**time (seconds)**			
	$t_{0.5}$	$t_{0.7}$	$t_{0.9}$	$t_{1.1}$	$t_{0.5}$	$t_{0.7}$	$t_{0.9}$	$t_{1.1}$	$t_{0.5}$	$t_{0.7}$	$t_{0.9}$	$t_{1.1}$
BPI2011	6695	1378	37	0	123	126	100	68	1697.03	191.62	31.25	24.96
BPI2012	12	9	4	0	20	19	18	15	5.88	6.02	6.79	6.06
BPI2014	15	3	0	0	8	8	8	8	42.41	41.54	43.97	43.36
BPI2015	4508	2194	239	0	616	687	391	165	167.73	161.41	49.95	18.36
BPI2016	2767	575	17	0	22	25	22	10	188.23	150.41	39.63	9.62
BPI2017	10	3	0	0	20	20	20	20	12.20	8.02	8.15	8.14

$\langle b, c1, c2, c, d, b3\rangle$, $\langle b, c2, c1, c, d, b2\rangle$, $\langle b, c1, c2, c, d, b2\rangle$, $\langle b, c1, c2, c, d, b1, e, f, f, g, b2\rangle$, and $\langle b, c2, c1, c, d, d, d, b1, e, g, b2\rangle$. More complex long-term dependencies can be identified in this log. The log contains a long-term concurrency relation between $a1$ and $a2$ after a. Moreover, there is a long-term choice between the activities $b2$ and $b3$ after b. In case $b2$ is selected, a sequence of activities starting with $b1$ can be executed before $b2$.

Figure 4b shows the causal graph discovered for L_3 using the parameters $t_{LD} = \alpha = 0.5$ and $t_{R_S} = 0.3$. The long-term concurrency relation between $a1$ and $a2$ after a is captured by the long-term dependency edges $(a,\ a1)$ and $(a,\ a2)$. The long-term choice between the activities $b2$ and $b3$ after b is captured by the long-term dependency edges $(b,\ b2)$ and $(b,\ b3)$. An additional long-term dependency edge $(b,\ b1)$ is discovered for representing the case where an optional sequence of activities starting with $b1$ is executed after b.

4 Evaluation

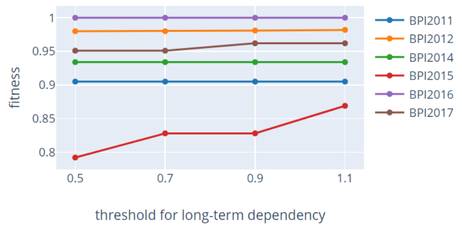
In this section, we evaluate the extended causal graph miner based on real-life event logs[3]. We evaluate the effect of changing the value of the parameter t_{LD} on the time performance of the hybrid miner (Section 4.1) and the quality of the discovered models (Section 4.2).

4.1 Time Performance

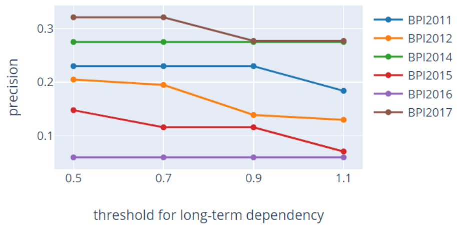
In this section, we use the same six BPI Challenge data sets [8, 9, 10, 11, 2, 12] used to evaluate the initial version of the hybrid miner in [7]. We use the parameters $t_{R_S} = w = 0.5$. For the long-term dependency threshold (t_{LD}), we test the values 0.5, 0.7, 0.9, and 1.1 ($t_{LD} = 1.1$ means deactivating the detection of long-term dependencies). We use the plugin "Extended Hybrid Petri Net Miner" in ProM [13] to discover hybrid Petri nets based on the discovered causal graphs. The parameter $t_{eval} = 0.6$ is used in the second discovery step.

The results of the evaluation are shown in Table 1. For each case, we report the number of discovered long-term dependency edges in the first discovery step, the number of discovered places in the second discovery step, and the time needed to discover the models. The results show that, in general, decreasing the value of t_{LD} increases the time required to discover the models. For instance,

[3] A new plugin "Extended Causal Graph Miner" has been implemented in ProM [13] to support the approach introduced in this paper.



(a) Fitness.



(b) Precision.

Fig. 5: Conformance checking results of the evaluation of the parameter t_{LD}.

decreasing the value of t_{LD} from 0.9 to 0.7 for BPI2016 increased the time from 39.63 seconds to 150.41 seconds. This behavior was expected because adding more long-term dependency edges leads to the generation of more places in the second discovery step, and the evaluation of these places is a time-consuming step. However, we observe that increasing the number of long-term dependency edges does not necessarily mean increasing the number of places in the final hybrid Petri net. For example, decreasing the value of t_{LD} from 0.7 to 0.5 for BPI2011 increased the number of long-term dependency edges from 1378 to 6695, but it decreased the number of places from 126 to 123. This might be due to the replacement of a group of places by a larger place covering all of their underlying dependencies.

We observe that many long-term dependencies are not transformed into places in the second discovery step. This behavior was expected as the core idea of the hybrid miner is to create models with both formal and informal parts. For instance, let us consider the log BPI2014. Decreasing the value of t_{LD} from 0.9 to 0.5 generated 15 long-term dependency edges, but no places are added in the second discovery step based on them.

4.2 Qualitative Evaluation

We evaluated the quality of the discovered hybrid Petri nets in Section 4.1 by applying conformance checking techniques[4][1]. We were not able to get conformance checking results for many of the models due to out-of-memory exceptions. Therefore, we repeat the experiment using the same settings but after preprocessing the event logs by filtering them. For BPI2016, we filter out activities that are present in less than 50% of traces; for BPI2017, we filer out trace variants that cover less than 1% of traces; for the other logs (BPI2011, BPI2012, BPI2014, and BPI2015), we filter out trace variants that have an absolute frequency of 1.

We omit the time results of the second experiment because all models were discovered in less than a second. The conformance checking results are shown in Figure 5. For BPI2014 and BPI2016, no places were discovered based on long-term dependencies, and the quality of the discovered models did not change, therefore. For the other four logs (BPI2011, BPI2012, BPI2015, and BPI2017), decreasing the value of t_{LD} improved the precision of the discovered models. For BPI2011, enabling the detection of long-term dependencies increased the precision from 0.184 to 0.230 without affecting the fitness. However, we observe a decrease in fitness after decreasing t_{LD} for other cases. For instance, decreasing the value of t_{LD} from 0.9 to 0.7 for BPI2017 decreased the fitness from 0.962

[4] In order to apply conformance checking techniques, hybrid Petri nets are transformed into standard Petri nets by simply removing all informal arcs.

to 0.951 and increased the precision from 0.277 to 0.321. This behavior was expected because detecting long-term dependencies leads to the generation of additional places, and adding more places often lowers the fitness of the model as more traces become non-fitting.

Summary. The time performance evaluation (Section 4.1) shows that the detection of long-term dependencies generates additional time costs. The qualitative evaluation (Section 4.2) shows that detecting long-term dependencies improves the precision of the models, but it can reduce the fitness as more places are discovered. We assess these differences in time and fitness to be acceptable as the goal of detecting long-term dependencies is to reduce the representational bias of the hybrid miner and produce more precise models. Based on our evaluation, we recommend using high values for the long-term dependency parameter (t_{LD}) in order to avoid high time costs and achieve a trade-off between fitness, precision, and simplicity.

5 Related Work

In [6], van der Aalst presented a wide range of process discovery approaches. The heuristic miner [6, 15] is a two-step discovery approach that produces an informal dependency graph in the first discovery step, and it uses this informal model to generate a formal causal net in the second discovery step. The causal graph created by the hybrid miner is inspired by the first discovery step of the heuristic miner. The idea of combining formal and informal models in process discovery and the notation of hybrid Petri nets were first introduced in [7]. The initial hybrid miner [7] was inspired by the idea of modeling vagueness suggested in [3, 4]. In [5, 17], other types of hybrid process models are defined by combined declarative and imperative modeling notations.

In this paper, we introduce an approach for improving causal graphs to detect and filter long-term dependencies. The discovery approach introduced in [14] also mines for long-term dependencies as a post-processing step. However, the approach in [14] is restricted to simple long-term dependencies as it assumes an equal frequency for any pair of activities of a long-term dependency relation. Our approach allows for the detection of more complex long-term dependency structures. In [16], another approach for detecting non-free-choice constructs is proposed. The main difference between this approach and our approach is that we mine for long-term dependencies in the first discovery step; we generate a causal graph that only contains the filtered set of long-term dependencies. In [16], redundant implicit dependencies are dynamically eliminated while generating the places of the final Petri net.

6 Conclusion

Process discovery is one of the main branches of process mining. Based on an event log, a process discovery approach generates a process model that captures the behavior recorded in the log. The hybrid miner is a two-step process discovery approach that combines formal Petri nets with an informal representation of vague structures. In the first discovery step, an informal causal graph is discovered, and this graph is used to generate candidate places in the second discovery step. In this paper, we introduced an extended version of the first discovery step. The new causal graph discovery approach enables the detection of long-term dependencies. This helps to reduce the representational bias of the hybrid miner as the additional long-term dependencies are used to generate candidate places in the second discovery step. We implemented the extended version of the causal graph miner in ProM [13] and we evaluated it using real-life event logs.

We propose multiple ideas for future work. Although enabling the detection of long-term dependencies reduces the representational bias of the hybrid miner, it is still not able to model some simple structures. For instance, the hybrid miner is not able to model any structures that require using silent or duplicate transitions. Another important topic for future work is to investigate the time performance of the hybrid miner and to propose solutions for speeding up the generation and evaluation of candidate place in the second discovery step. We also suggest tailoring process conformance checking techniques to support hybrid Petri nets and defining new metrics for evaluating the quality of hybrid Petri nets. Finally, the idea of combining formal and informal models is not restricted to the discovery of hybrid Petri nets. This idea can be applied to other types of process models to discover new types of hybrid models.

References

1. Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking - Relating Processes and Models.* Springer, 2018.
2. M. Dees and Boudewijn van Dongen. BPI Challenge 2016: Clicks Logged In. 2016.
3. Thomas Herrmann, Marcel Hoffmann, Kai-Uwe Loser, and Klaus Moysich. Semistructured models are surprisingly useful for user-centered design. In *Proceedings of the 4th International Conference on Designing Cooperative Systems*, pages 159–174. IOS Press, 2000.
4. Thomas Herrmann and Kai-Uwe Loser. Vagueness in models of socio-technical systems. *Behav. Inf. Technol.*, 18(5):313–323, 1999.
5. Hajo A. Reijers, Tijs Slaats, and Christian Stahl. Declarative modeling–an academic dream or the future for bpm? In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management - 11th International Conference*, pages 307–322. Springer Berlin Heidelberg, 2013.
6. Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition.* Springer, 2016.
7. Wil M. P. van der Aalst, Riccardo De Masellis, Chiara Di Francescomarino, and Chiara Ghidini. Learning Hybrid Process Models from Events - Process Discovery Without Faking Confidence. In Josep Carmona, Gregor Engels, and Akhil Kumar, editors, *Business Process Management - 15th International Conference*, volume 10445 of *Lecture Notes in Computer Science*, pages 59–76. Springer, 2017.
8. Boudewijn van Dongen. Real-life event logs - Hospital log. 2011.
9. Boudewijn van Dongen. BPI Challenge 2012. 2012.
10. Boudewijn van Dongen. BPI Challenge 2014. 2014.
11. Boudewijn van Dongen. BPI Challenge 2015. 2015.
12. Boudewijn van Dongen. BPI Challenge 2017. 2017.
13. Boudewijn F. van Dongen, Ana Karla A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and Wil M. P. van der Aalst. The Prom Framework: A New Era in Process Mining Tool Support. In Gianfranco Ciardo and Philippe Darondeau, editors, *Applications and Theory of Petri Nets 2005, 26th International Conference, Proceedings*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005.
14. A. J. M. M. Weijters and Joel Tiago S. Ribeiro. Flexible Heuristics Miner (FHM). In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011*, pages 310–317. IEEE, 2011.
15. A. J. M. M. Weijters and Wil M. P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integr. Comput. Aided Eng.*, 10(2):151–162, 2003.
16. Lijie Wen, Wil M. P. van der Aalst, Jianmin Wang, and Jiaguang Sun. Mining process models with non-free-choice constructs. *Data Min. Knowl. Discov.*, 15(2):145–180, 2007.
17. Michael Westergaard and Tijs Slaats. Mixing paradigms for more comprehensible models. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management - 11th International Conference*, volume 8094 of *Lecture Notes in Computer Science*, pages 283–290. Springer, 2013.