Discovering Hybrid Process Models With Bounds On Time and Complexity

When to be formal and when not?

Wil M.P. van der Aalst[a], Riccardo De Masellis[b], Chiara Di Francescomarino[c], Chiara Ghidini[b], Humam Kourani[d]

[a]*Lehrstuhl für Informatik 9 / Process and Data Science, RWTH Aachen University, D-52056 Aachen, Germany*
[b]*FBK-IRST, Via Sommarive 18, 38050 Trento, Italy*
[c]*University of Trento, via Sommarive 9, 38123 Trento, Italy*
[d]*Fraunhofer FIT, Schloss Birlinghoven, 53757 Sankt Augustin, Germany*

Abstract

Discovering process models from event data is a highly relevant, but also a notoriously difficult, problem. Therefore, it is unsurprising that the biggest share of process mining research is devoted to process discovery. While techniques reported in scientific literature tend to produce process models that are formal, i.e., which mathematically describe the possible behaviors, commercial process mining tools return informal models (merely a "picture" not allowing for any form of formal reasoning). *Hybrid process models* aim at combining the best of both worlds: they capture behavior that is strongly supported by data and that can be used for formal reasoning, as well as behavior that cannot be represented in clear-cut process constructs or that does not have enough evidence in the data. This paper presents an approach for discovering *hybrid Petri nets*, which, unlike existing techniques, produces models that have both formal and semi-formal constructs so that even if the behavior in the data is noisy and irregular or it does not fit predefined constructs, causal relationships are still captured. Our evaluation demonstrates the advantages of combining such "deliberate vagueness" with formal guarantees. The ideas presented here are fairly general, and can serve as a foundation for other, new hybrid discovery techniques.

Keywords:
Process mining, Process discovery, Petri nets, BPM

1. Introduction

Process mining approaches are both data-driven and process-centric [1]. The starting point is generally event data, which can be used to discover models or to check compliance with respect to a normative process, but once the events are coupled to process models, a range of analysis techniques are enabled. For example, some process mining techniques diagnose bottlenecks based on the evidence in the data [2, 3] while others use the discovered models to predict problems (e.g., deviations or delays) and to recommend actions [4, 5]. All these techniques rely on high-quality process models that are well-aligned with the event data [4]. However, discovering such models is extremely challenging in the presence of irregular, noisy behavior.

The uptake of process mining is reflected by the growing number of commercial process mining tools available today. There are over 40 commercial products enabling process mining, e.g., Celonis, Signavio/SAP, myInvenio/IBM, Disco, Minit/Microsoft, ProcessGold/UiPath, LanaLabs/Appian, Apromore and many others, and all support process discovery, as it can be used to improve compliance and performance problems [6]. These commercial tools are based on variants of techniques like the heuristic miner [7] and the fuzzy miner [8] and typically produce filtered Directly Follows Graphs (DFGs) [1]. They are not able to discover concurrency and they produce informal (also referred to as semi-formal) process models, that is, models without *formal semantics*. Such informal ("boxes and arrows") models provide valuable insights, but cannot be used to draw reliable conclusions. For example, they cannot be used as a *classifier* for traces. Classifying traces into *fitting* (behavior allowed by the model) and *non-fitting* (not possible according to the model) is however important for more advanced types of process mining. Therefore, most discovery algorithms described in the literature (e.g., the α-algorithm [9], the state-based region approaches [10, 11, 12, 13, 13], the language-based region approaches [14, 15, 16, 17, 18, 19], the declarative approaches [20, 21, 22, 23, 24, 25], and the inductive mining approaches [26, 27, 28]) produce formal models (Petri nets, transition systems, automata, temporal logics, process trees, etc.) having clear semantics and being able to uncover concurrency.

At first sight, is seems appealing to discover formal models rather than informal models. However:

- Formal models act as binary classifiers: traces are fitting or non-fitting. For real-life processes this is often not so clear-cut. It may be impossible to capture behavior using the process model constructs provided (e.g., places

in Petri nets or gateways in BPMN) or the data may be inconclusive and may not justify formalization. The model capturing 80 percent of all traces may be simple and more valuable than the model that allows for all outliers and deviations seen in the event log. However, we *cannot* simply discard all data not directly contributing to undisputed formal model elements. Hence, "vagueness" may be desirable to show additional relationships.

- Formal models may be hard to understand. End-users need to be able to interpret process mining results: Petri nets with smartly constructed places and BPMN with many gateways are quickly perceived as too complex. Underfitting models, namely models that allow for the behavior observed in the event log but also for much more behavior, may indeed be simpler to observe. However, they fail to capture knowledge contained in the data by allowing much more behavior. On the other hand, overfitting models, that is, models that only allow for the behaviors seen in the logs, are more complex and may lead to conclusions that are valid only for the observed data.

- Commercial process mining tools need to be able to handle logs with millions of events and still be used in an interactive manner. Many of the more sophisticated discovery algorithms producing formal models (e.g., region-based approaches [14, 13, 18]) do not scale well.

This explains why commercial process mining tools resort to informal models. These tools are scalable, simple, and easy to use. However, a *precise* interpretation of the generated informal model is often impossible as, informal models do not provide clear semantics for splitting and joining points. When an activity has multiple outgoing arcs, i.e., multiple preceding activities, one would like to know whether these are concurrent or in a choice relation. Which combinations of output arcs can be combined? Showing frequencies on nodes (activities) and arcs may further add to the confusion when "numbers do not add up". With no clear semantics and formal guarantees, the process model remains a "picture" and will be distrusted.

In [29], we proposed *hybrid process models* as a way to combine the best of both worlds. Such models show informal dependencies (like in commercial tools) that are deliberately vague and at the same time they provide formal semantics for the parts that are clear-cut. Whenever there is enough structure and evidence in the data, explicit routing constructs are used; if, instead, dependencies are weak or too complex, then they are depicted in an informal manner, rather than being left out. The technique presented in this paper improves the earlier approach in

two ways. First of all, we managed to *significantly improve the performance of the earlier approach* by *(i)* ordering the set of candidate places; *(ii)* pruning the set of candidate places with the introduction of new notions of conflict and exclusion and *(iii)* quickly removing candidate places that do not meet the preset quality criteria in a safe manner. Second, we provide *more control* over the hybrid process model that is generated. Moreover, the approach incorporating all the new optimizations has been reimplemented in ProM. The new version of the ProM package *HybridMiner* provides the plug-ins *Causal Graph Miner* and *Hybrid Petri Net Miner* corresponding to the two main phases of our approach. The software is open source and can be obtained from `www.promtools.org`. Finally, we provide a *broad* evaluation of the new approach that takes into account different perspectives and more datasets.

We use *hybrid Petri nets*, a new class for Petri nets with informal annotations, as a concrete representation of hybrid process models. However, the ideas, concepts, and algorithms are generic and could also be used in the context of BPMN, UML activity diagrams, etc. In a hybrid Petri net, there are three types of arcs: normal arcs, sure arcs, and unsure arcs. Normal arcs are traditional, Petri net arcs; sure arcs should be interpreted as causal relationships between activities that cannot be expressed (easily) in a formal manner, viz., in terms of a Petri net place and finally, unsure arcs are suspected causal relationships that are too weak to justify a place connecting activities. We note that, when all causality relationships among activities are clear and can be expressed in terms of places, a hybrid Petri net looks exactly as a Petri net. Our *discovery technique* has two phases. First, we discover a *causal graph* from the event log. Based on different (threshold) parameters we scan the event log for possible causalities. In the second phase, we learn places based on explicit quality criteria, which are interpreted in a precise manner and have guaranteed quality. Causal relations that cannot or should not be expressed in terms of places are added as sure or unsure arcs. Once a hybrid Petri net is discovered, it can be used as a starting point for other types of process mining. We acknowledge a few limitations of our approach: it does not allow for label duplication and silent transitions and it requires the setting of some parameters, e.g., the threshold parameters. However, the latter is mitigated by the fact that some default values are provided and that an evaluation on the impact of the parameter values on the performance is carried out, so as to provide the users with some guidelines on parameter configuration.

Note that our objective is much broader: We argue that hybrid process models are useful and combine *simplicity, vagueness, and scalability with partly formal models that allow for reasoning and provide formal guarantees*. The concrete

discovery technique and extensive evaluations presented are intended as an illustration of this broader vision. Moreover, the paper demonstrates that other approaches may provide misleading results or return models that unexpectedly do not allow for any of the traces seen in the data.

The remainder is organized as follows. In Section 2, we motivate our work by discussing the weaknesses of existing techniques that produce either formal models or informal models. Section 3 introduces event logs and traditional Petri nets, while Section 4 presents causal graphs along with some metrics which will be used in the evaluation. Section 5 defines hybrid Petri nets. Section 6 first discusses ways of measuring the quality of a place in relation to an event log and elaborates on problems related to conflicting places, and then introduces our two-phase discovery algorithm. Section 7 describes the ProM plug-ins developed to support the discovery of hybrid process models. Section 8 evaluates the approach in four different ways. Section 9 discusses related work and Section 10 concludes the paper.

2. Problem Analysis

In this section, we discuss problems related to formal and informal models, and we define the requirements for an ideal hybrid process model. To motivate our contribution, an example of a process is provided.

2.1. Problems Related to Informal Models

Commercial process mining tools like Celonis, Signavio/SAP, Disco, myInvenio/IBM, Minit/Microsoft, ProcessGold/UiPath, etc. produce informal models that can be described as "boxes and arrows". These tools are inspired by the fuzzy miner [8], which was the first to provide sliders allowing the user to seamlessly simplify the model. Whereas the fuzzy miner provided quite some control over the creation of arcs, commercial systems simplified matters by providing two non-configurable frequency-based sliders. The discovered models do not show the split and join behavior (AND, XOR, or OR) in the process model. In some cases the tools are able to discover concurrency, but this is not shown. Sometimes concurrency is only detected when activities overlap (Disco) or additional information is given (Celonis), but this is not exposed to the user. To get an interpretable model, there is often the possibility to simply show the *Directly Follows Graph* (DFG henceforth). In this graph, two activities are connected if and only if one is followed by the other somewhere in the event log. The number on the arc connecting activities a and b then indicates how often a was followed by b.

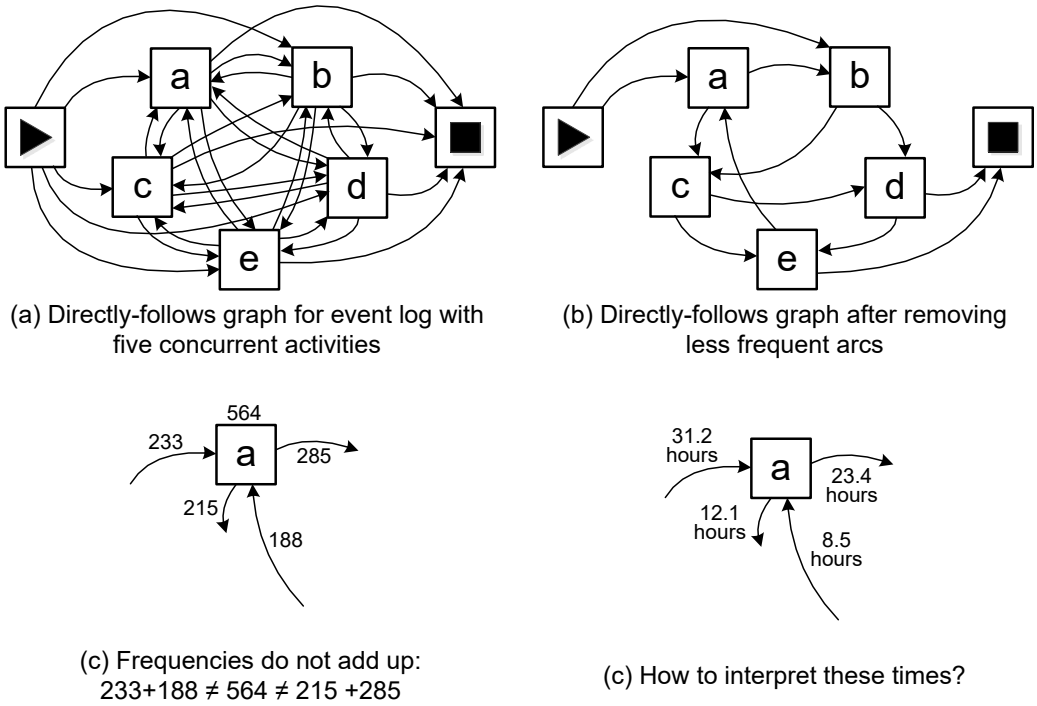


Figure 1: Problems encountered when discovering a process with five parallel activities using a commercial tool unable to discover concurrency.

The DFG is easy to generate and seems easy to interpret. However, whenever a set of activities does not happen in a fixed order, the model becomes confusing: the DFG suggests loops whereas all activities are executed only once and the usage of frequency-based filtering only adds to the confusion, as some loops might remain while other possible paths disappear.

Figure 1 sketches the problem more in detail. Let us consider an event log L with five parallel activities (a, b, c, d, and e) where all possible interleavings are observed, so that for instance $\langle a, b, c, d, e\rangle \in L$, but also $\langle b, d, e, a, c\rangle \in L$ as well as any other sequence. The DFG without frequency-based filtering is shown in Figure 1(a). The graph suggests loops whereas each activity is only executed once. Using frequency-based filtering, we may remove the less frequent paths resulting in the model shown in Figure 1(b), which shows the dominant behavior but also triggers many questions. When showing the frequencies of activities and direct successions, numbers do not add up after filtering. In Figure 1(c) activity a was executed 564 times. However, the numbers of the input and output arcs do not add

up. When average delays are shown in a DFG they refer to the mean time between two activities under the condition that the activities directly followed each other. Based on Figure 1(c) one could think that a is executed 31.2 hours after the start of the case and b is executed 23.4 hours after a (on average). However, this is not the case: a is executed 31.2 hours (on average) after the start under the condition that a was the first activity, but when a was not the first, these times could have been very different. Also the real average time between a and b may be longer or shorter than 23.4 hours.

Figure 1 illustrates that simple "boxes and arrows" diagrams may cause confusion and conceal the actual process (which is very structured and simple). Moreover, frequencies and times need to be interpreted carefully. The numbers in Figure 1 are not wrong, but can be misinterpreted easily.

2.2. Problems Related to Formal Models

The majority of the discovery algorithms described in the literature (e.g., the α-algorithm [9], the region-based approaches [14, 13, 18], and the inductive mining approaches [26, 27, 28]) produce formal models able to capture sequences, choices, concurrency, and loops. These can be transition systems, different flavors of Petri nets, automata, process trees, BPMN models, etc. Formal models may also be declarative. A model is formal if it is able to classify traces into fitting (i.e., part of the language) and non-fitting. Figure 1(b) could be interpreted as a formal model. However, it is not intended as such.

To explain these challenges, let us first consider a very simple event log $L_1 = [\langle a, b, d, e, f\rangle^{100}, \langle a, c, e, d, f\rangle^{100}]$. L_1 contains 200 cases, each composed of 5 events (1000 events in total). For 100 cases the following sequence is executed $\langle a, b, d, e, f\rangle$ and for the other 100 cases $\langle a, c, e, d, f\rangle$ (more precise definitions of multisets, logs and their notations will be presented in Section 3.1). Figure 2(a) shows the Petri net discovered by the α-algorithm [9]. The model is deadlocking (f needs to synchronize but will never be able to do so) and none of the original cases fits the model. Figure 2(b) shows the process model discovered by the inductive miner [26]. This model is underfitting and allows for unseen traces $\langle a, c, d, e, f\rangle$ and $\langle a, b, e, d, f\rangle$. Figure 2(c) shows the process model discovered by the ILP miner [18]. Also, this model allows for too much behavior. It is also possible to use a process discovery technique that allows for multiple transitions having the same label (e.g., state-based regions with label splitting [10, 30]). Such an approach will be able to discover a model only allowing for the two traces observed (see Figure 2(d)).

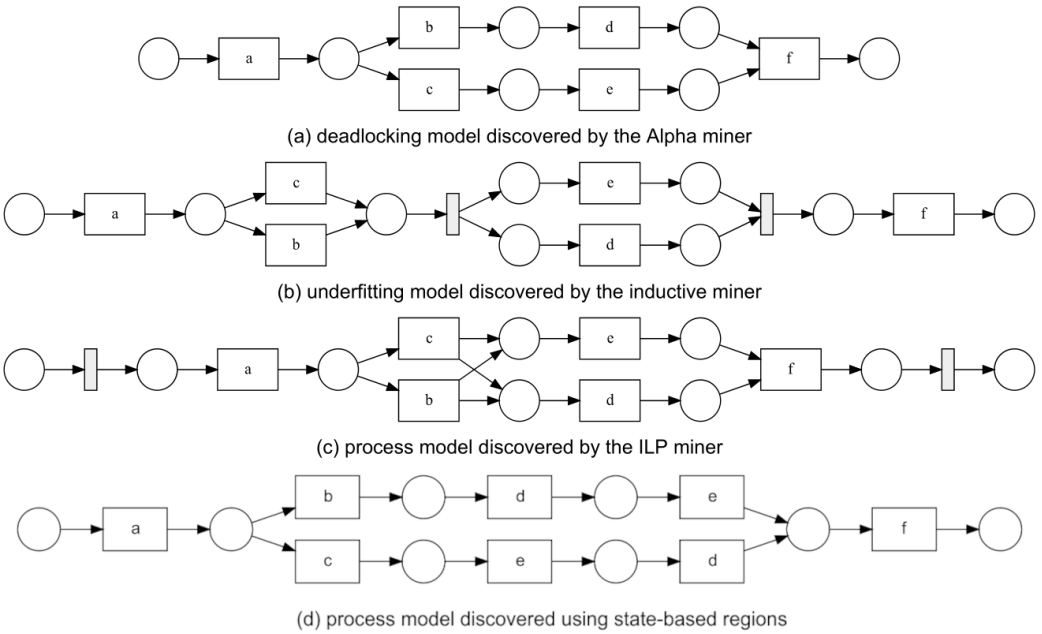


Figure 2: A few formal process models (represented in terms of Petri nets) discovered for the event log $L_1 = [\langle a, b, d, e, f \rangle^{100}, \langle a, c, e, d, f \rangle^{100}]$.

Event log L_1 has a lot of structure. Now we add a bit of exceptional behavior: we add 10 traces (i.e., 5%) where one of the events is replaced by an x event. This may have a dramatic effect on the process models. Figure 3(a) shows the Petri net discovered by the α-algorithm. The model is still deadlocking and unable to explain any of the observed traces. Figure 3(b) shows the process model discovered by the inductive miner. We used a threshold of 0, ensuring that the model captures all observed behavior. It does, but the resulting model is severely underfitting. All activities can be skipped, e.g., the empty trace $\langle\ \rangle$ is a possible behavior. The process model discovered by the ILP miner shown in Figure 3(c) using the same (standard) settings as used for Figure 2(c) allows not only to capture the behavior of the event log but it also allows for any behavior. Figure 3(d) shows the process model discovered using state-based regions with label splitting. This model captures the behavior seen in the event log in a smart way, but is rather complicated and could be considered as overfitting. A single event can dramatically change the model.

Figures 2 and 3 show the problems of purely formal approaches. First of all, such approaches need to make a binary decision with respect to possible traces.

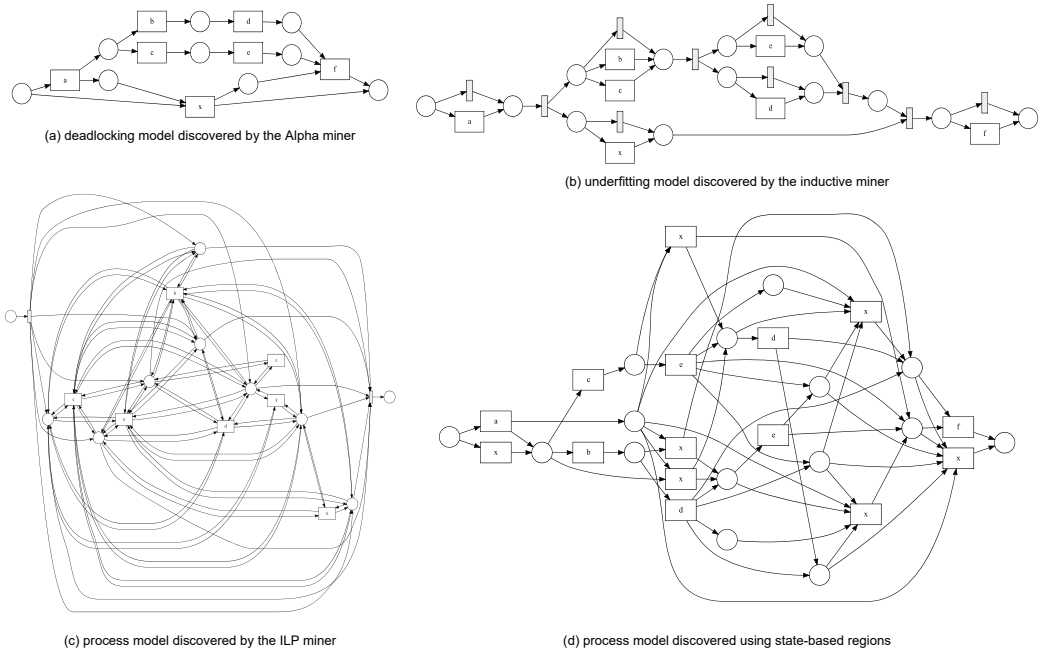


Figure 3: Formal process models when adding 10 noisy cases where one "regular" event was replaced by an exceptional event x.

A formal model needs to make a binary decision whether a certain behavior is included or not, that is whether a trace fits or not. Second, any discovery approach is limited by its *representational bias*. The representational bias refers to the class of models than can be discovered by the considered discovery approach.[1] For example, the α-algorithm can only discover Petri nets where all transitions have a unique and visible label. The same holds for the ILP miner. The inductive miner can only discover block-structured models where all activities are unique. Therefore, it cannot discover the dependency between b and c and the ordering of d and e in Figure 2(b). Changing the representational bias can, at best, only provide a partial solution. Adding more sophisticated modeling constructs will lead to models that are difficult to understand and also difficult to discover. Moreover, real-life behavior will seldom perfectly match the representational bias of some formal modeling language. Therefore, one has to accept that behavior cannot be

[1]Please note that here we are not necessarily stating that there is no Petri net able to capture the behavior in the example, but rather that the representational bias of the approach does not allow for discovering it.

fully captured in a model that classifies traces into fitting or non-fitting.

The presence of exceptional behavior (like the randomly inserted activity x in Figure 3) may lead to degenerate models because the discovery techniques try to straightjacket erratic behavior in their representational bias. Most of the more recent mining algorithms provide the option to create a fitting model. For example, the inductive model guarantees that the discovered model is able to generate the event log when the noise threshold is set to 0. However, this may lead to severely underfitting models that allow for behavior unrelated to the observed behavior and besides, such guarantees are only provided for such extreme settings and do not apply when the approach is used in practice. Later, we will show examples where also the inductive miner does not allow for any of the traces seen in the event log.

In short, existing formal approaches suffer from at least one of the following problems:

- Discovered process models may be deadlocking and/or not allow for the majority of the behavior seen (Figures 2(a) and Figure 3(a)).
- Discovered process models may be underfitting the observed behavior when the mainstream behavior is outside the representational bias or because of erratic/infrequent behavior (Figures 2(b), 2(c), 3(b), and 3(c)).
- Discovered process models may be overfitting the observed behavior when the mainstream behavior is outside the representational bias or because of erratic/infrequent behavior (Figure 3(d)).

All have the problem that the behavior that cannot be captured due to the representational bias is not represented.

2.3. Requirements

Ideally, a discovery approach has the following properties:

R1. What can be formalized should be formalized. Whenever behavior can be captured using a formal construct it should not be left informal. What is formalized should have a clear and unambiguous interpretation.

R2. What is formalized should be supported by formal guarantees that can be interpreted by the user. Guarantees should hold for all parameter settings and not just "in the limit".

R3. What cannot be formalized should be left informal. Behavior that cannot be captured formally (due to limitations of the representational bias or erratic/infrequent behavior), should not lead to degenerate or incorrect models.

R4. What cannot be formalized should not be left out and captured in a different way. The model should provide information on behavior that was not captured formally.

In this paper, we propose to use *hybrid process models* to address the above requirements. In the evaluation, we return to these requirements.

2.4. Motivating Example

To better understand the pitfalls of existing process discovery techniques, let us consider an event log related to an order handling process. Figure 4 shows a trace variant representation of our example event log (i.e., a representation of unique activity sequences with their frequencies). This event log consists of 12 666 cases that can be grouped into nine unique trace variants.

All five models in Figure 5 and Figure 6 have been produced for this event log using state-of-the-art process discovery approaches. Models (d) and (e) were created using the commercial process mining tool *Disco* (Fluxicon) using different settings. These models are informal. Model (d) shows only the most frequent paths and model (e) shows all possible paths. For such informal models, it is impossible to determine the exact nature of splits and joins. Commercial tools have problems dealing with loops and concurrency. For example, for all cases, activities *make delivery* and *confirm payment* happened at most once, but not in a fixed order. However, these concurrent activities are put into a loop in models (d) and (e). This problem is not specific to *Disco* or this event log: all commercial tools suffer from this problem when using informal graphs.

Models (a), (b), and (c) are expressed in terms of Petri nets and have formal semantics. Models (b) and (c) were created using the inductive miner (IMf [27]) with different settings for the noise threshold (0.0 respectively 0.2). Model (b) is underfitting. For instance, this model allows for instances where the ordered items are delivered although the payment is skipped. Model (c) is neither fitting nor precise; i.e., this model only allows for the mainstream behavior observed in the log, and it allows for additional behavior not observed in the log. For example, cases with multiple reminders and cases where the payment is done before sending the invoice do not fit the model, while cases where the order is canceled after the

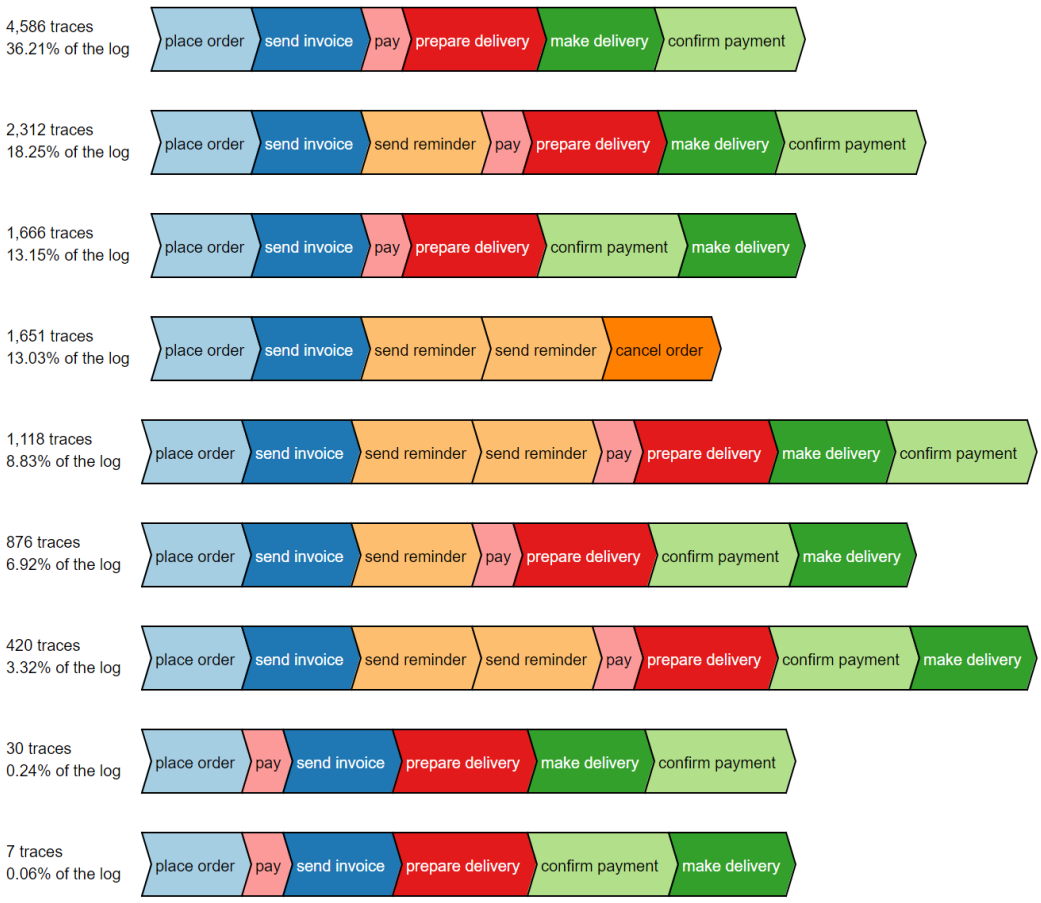
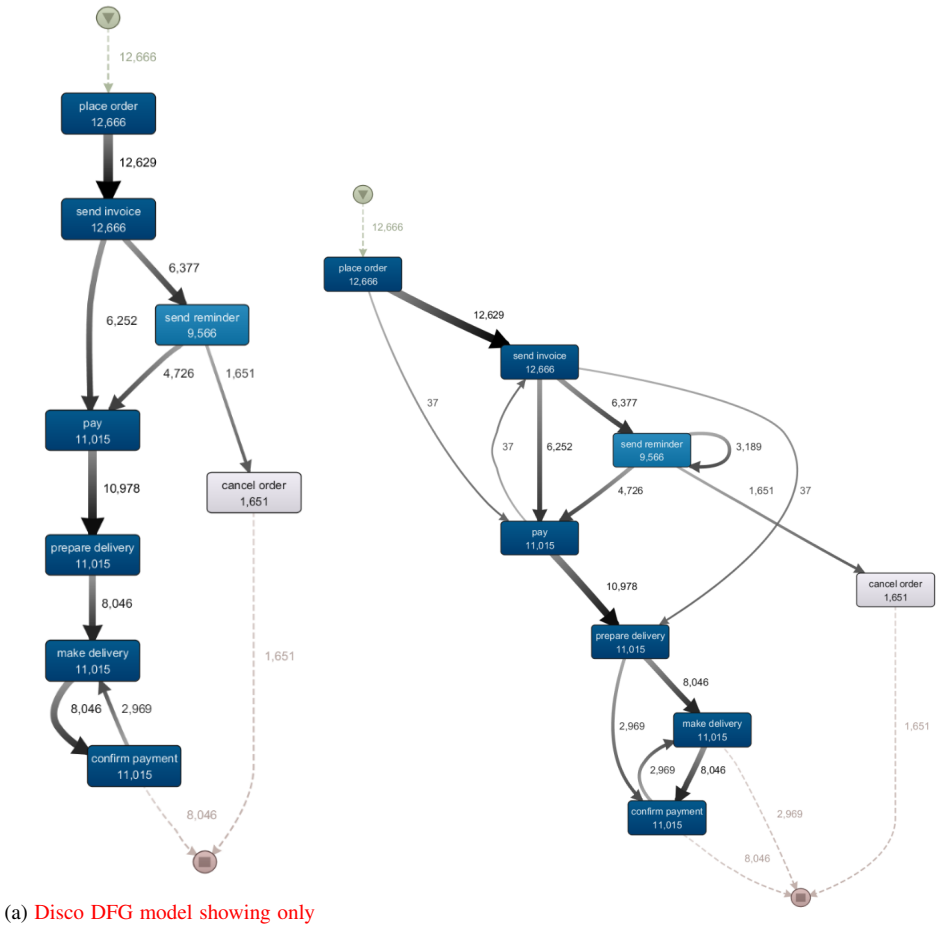


Figure 4: Order handling event log.

payment are allowed. Model (a) was created using the ILP miner with default settings. This model is able to precisely describe the choice between *pay* and *cancel order*; however, it still suffers from some issues. For instance, traces with *confirm payment* before *make delivery* do not fit the model, and it allows for skipping *send reminder* and directly executing *cancel order* after *send invoice*. Moreover, model (a) contains a much higher number of arcs compared to the other models. For larger event logs having more activities and low-frequent paths, the ILP miner is not able to produce meaningful models (the approach becomes intractable and/or produces incomprehensible models).

We introduce *hybrid Petri nets* to combine the models in Figure 5 and the ones in Figure 6; i.e., we use formal semantics when the behavior is clear and easy to

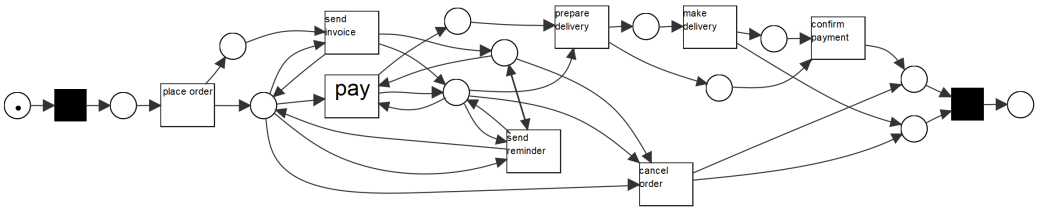


(a) Disco DFG model showing only the most frequent paths

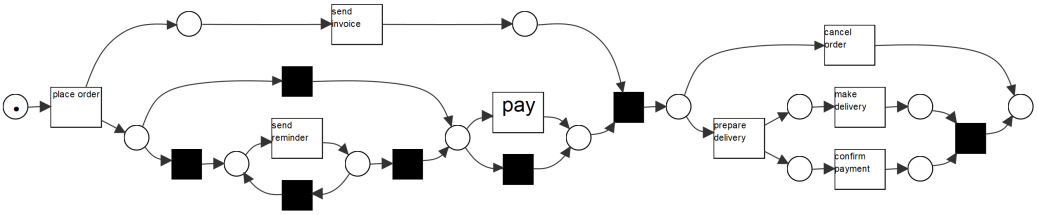
(b) Disco DFG model showing also the infrequent paths

Figure 5: Process models discovered by Disco for an event log recorded for 12 666 orders.

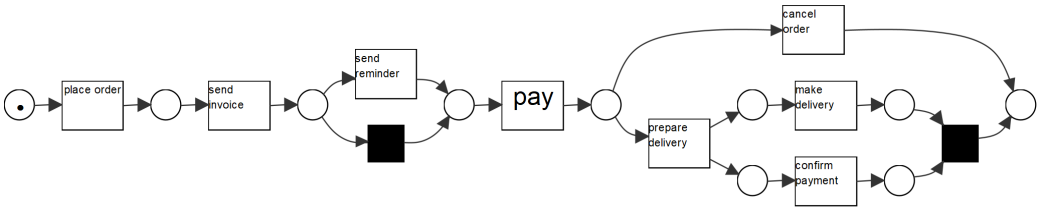
express and we resort to informal annotations when things are blurry or inexact. Figure 7 shows a hybrid Petri net modeling our example process. In this model, places represent dependencies that can be easily captured in a formal manner, e.g., to capture the sequential dependency between *place order* and *send invoice* and to capture the concurrency between *make delivery* and *confirm payment*. For vague structures where formal constraints cannot be justified, causal relations are depicted in the final hybrid Petri net as informal edges. A hybrid Petri net has two types of informal edges that directly connect transitions: *sure edges* represent strong causal dependencies between activities, and *uncertain edges* (dashed arcs) represent weak dependencies. For example, the sure edge (*send reminder* $\rightarrow$



(a) Accepting Petri net discovered by the ILP miner



(b) Underfitting process model discovered by the inductive miner



(c) Non-fitting process model discovered by the inductive miner

Figure 6: Process models discovered by ILP and inductive miner for an event log recorded for 12 666 orders (labels are not intended to be readable).

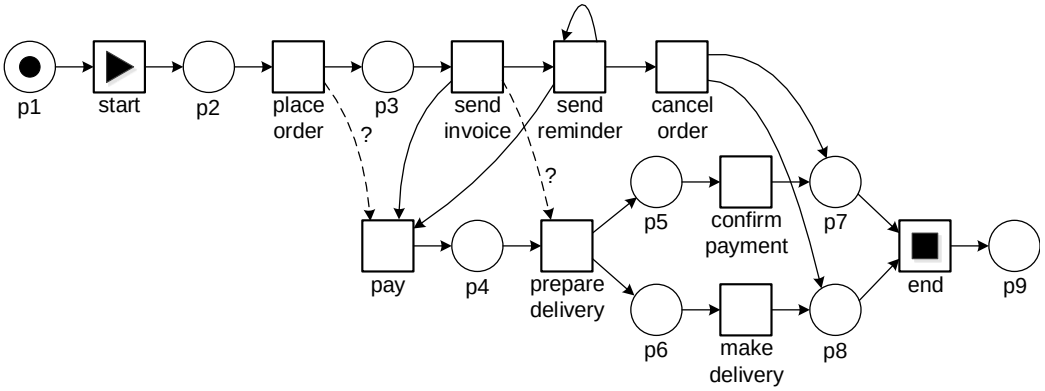


Figure 7: A hybrid system net discovered for the event log described in Figure 4.

cancel order) indicates a strong dependency between sending reminders and canceling the order; this dependency is strong since *cancel order* is always directly

preceded by *send reminder* in the event log, but this strong dependency cannot be easily captured using a formal place because *send reminder* is not always followed by *cancel order*. The uncertain edge (*place order* $\rightarrow$ *pay*) indicates a weak dependency between placing the order and paying; this covers the two infrequent trace variants where the customer directly pays after placing the order.

3. Event Logs and Petri Nets

In this section, we introduce basic concepts, including multisets, operations on sequences, event logs and Petri nets.

3.1. Preliminaries

$\mathcal{B}(A)$ is the set of all multisets over some set A. For some multiset $X \in \mathcal{B}(A)$, $X(a)$ denotes the number of times element $a \in A$ appears in X. Some examples: $X = [\,]$, $Y = [x^2, y]$, and $Z = [x^3, y^2, z]$ are multisets over $A = \{x, y, z\}$. X is the empty multiset, Y has three elements ($Y(x) = 2$, $Y(y) = 1$, and $Y(z) = 0$), and Z has six elements. Note that the ordering of elements is irrelevant.

With $\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in A^*$, that is the Kleene closure of the alphabet A, we denote a sequence over A; $\sigma(i) = a_i$ denotes the i-th element of the sequence; $|\sigma| = n$ denotes the length of σ and $dom(\sigma) = \{1, \ldots, |\sigma|\}$ is the domain of σ; $\langle\,\rangle$ is the empty sequence, i.e., $|\langle\,\rangle| = 0$ and $dom(\langle\,\rangle) = \emptyset$; $\sigma_1 \cdot \sigma_2$ is the concatenation of two sequences.

Let A be a set and $X \subseteq A$ one of its subsets. $\restriction_X$: $A^* \rightarrow X^*$ is a projection function, which we use with an infix notation, and is defined recursively: $\langle\,\rangle\restriction_X = \langle\,\rangle$ and for $\sigma \in A^*$ and $a \in A$: $(\langle a \rangle \cdot \sigma)\restriction_X = \sigma\restriction_X$ if $a \notin X$ and $(\langle a \rangle \cdot \sigma)\restriction_X = \langle a \rangle \cdot \sigma\restriction_X$ if $a \in X$. For example, $\langle a, b, a \rangle\restriction_{\{a,c\}} = \langle a, a \rangle$. Projection can also be applied to multisets of sequences, e.g., $[\langle a, b, a \rangle^5, \langle a, d, a \rangle^5, \langle a, c, e \rangle^3]\restriction_{\{a,c\}} = [\langle a, a \rangle^{10}, \langle a, c \rangle^3]$.

Later, we will also use summations over multisets, e.g., $\sum_{x \in [a,b,b,a,c]} x = 2a + 2b + c$. For example, if a trace appears multiple times in the log, this needs to be taken into account.

3.2. Event Logs

The starting point for process discovery is an event log where events are grouped into cases. Each case is represented by a trace, e.g., $\langle \triangleright, a, b, c, d, \square \rangle$.

Definition 1 (Event Log). *An* event log $L \in \mathcal{B}(A^*)$ *is a non-empty multiset of traces over some activity set A. A trace $\sigma \in L$ is a sequence of activities. There is*

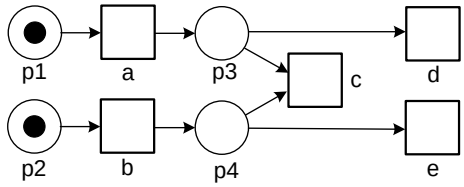


Figure 8: Small Petri net consisting of four places ($P = \{p1, p2, p3, p4\}$) and five transitions ($T = \{a, b, c, d, e\}$). In the initial marking $M_{init} = [p1, p2]$ there are two tokens. The final marking used for the corresponding system net is $M_{final} = [\,]$, i.e., the empty marking.

a special start activity $\triangleright$ *and a special end activity* $\square$*. We require that* $\{\triangleright, \square\} \subseteq A$ *and each trace* $\sigma \in L$ *has the structure* $\sigma = \langle \triangleright, a_1, a_2, \ldots, a_n, \square \rangle$ *and* $\{\triangleright, \square\} \cap \{a_1, a_2, \ldots, a_n\} = \emptyset$*.* $\mathcal{U}_L$ *is the set of all event logs satisfying these requirements.*

An event log captures the observed behavior that is used to learn a process model. An example log is $L_2 = [\langle \triangleright, a, b, c, d, \square \rangle^{45}, \langle \triangleright, a, c, b, d, \square \rangle^{35}, \langle \triangleright, a, e, d, \square \rangle^{20}]$ containing 100 traces and 580 events. In reality, each event has a timestamp and may have any number of additional attributes. For example, an event may refer to a customer, a product, the person executing the event, associated costs and so on. Here we abstract from these notions and simply represent an event by its activity name.

3.3. Petri Nets

A *Petri net* is a bipartite graph composed of places (represented by circles) and transitions (represented by squares). Figure 8 shows an abstract example to explain the basic concepts.

Definition 2 (Petri Net). *A Petri net is a tuple* $N = (P, T, F)$ *with* P *the set of places,* T *the set of transitions,* $P \cap T = \emptyset$*, and* $F \subseteq (P \times T) \cup (T \times P)$ *the flow relation.*

Transitions represent activities and places are added to model causal relations. $\bullet x = \{y \mid (y, x) \in F\}$ and $x\bullet = \{y \mid (x, y) \in F\}$ define input and output sets of places and transitions. Places can be used to causally connect transitions as is reflected by relation $\widehat{F}$: $(t_1, t_2) \in \widehat{F}$ if t_1 and t_2 are connected through a place p, i.e., $p \in t_1\bullet$ and $p \in \bullet t_2$. For the Petri net in Figure 8: $\widehat{F} = \{(a, c), (a, d), (b, c), (b, e)\}$.

Definition 3 ($\widehat{F}$). *Let* $N = (P, T, F)$ *be a Petri net.* $\widehat{F} = \{(t_1, t_2) \in T \times T \mid \exists_{p \in P} \{(t_1, p), (p, t_2)\} \subseteq F\}$ *are all pairs of transitions connected through places.*

The state of a Petri net, called *marking*, is a multiset of places indicating how many *tokens* each place contains. Tokens are shown as black dots inside places. Figure 8 shows the marking $[p1, p2]$.

Definition 4 (Marking). *Let $N = (P, T, F)$ be a Petri net. A marking M is a multiset of places, i.e., $M \in \mathcal{B}(P)$.*

A transition $t \in T$ is *enabled* in marking M of net N, denoted as $(N, M)[t\rangle$, if each of its input places ($p \in {\bullet}t$) contains at least one token. An enabled transition t may *fire*, i.e., one token is removed from each of the input places ($p \in {\bullet}t$) and one token is produced for each of the output places ($p \in t{\bullet}$). In Figure 8 both a and b are enabled and can fire independently.

$(N, M)[t\rangle(N, M')$ denotes that t is enabled in M and firing t results in marking M'. Let $\sigma = \langle t_1, t_2, \ldots, t_n\rangle \in T^*$ be a sequence of transitions, sometimes referred to as a *trace*. $(N, M)[\sigma\rangle(N, M')$ denotes that there is a set of markings $M_0, M_1, \ldots, M_n$ such that $M_0 = M$, $M_n = M'$, and $(N, M_i)[t_{i+1}\rangle(N, M_{i+1})$ for $0 \leq i < n$.

A *system net* has an initial and a final marking. The *behavior* of a system net corresponds to the set of traces starting in the initial marking M_{init} and ending in the final marking M_{final}.

Definition 5 (System Net Behavior). *A system net is a triplet $SN = (N, M_{init}, M_{final})$ where $N = (P, T, F)$ is a Petri net, $M_{init} \in \mathcal{B}(P)$ is the initial marking, and $M_{final} \in \mathcal{B}(P)$ is the final marking. $behav(SN) = \{\sigma \mid (N, M_{init})[\sigma\rangle(N, M_{final})\}$ is the set of traces possible according to the model.*

A system net SN classifies traces σ into *fitting* ($\sigma \in behav(SN)$) and *non-fitting* ($\sigma \notin behav(SN)$). For the system net shown in Figure 8 ($M_{init} = [p1, p2]$ and $M_{final} = [\,]$): $behav(SN) = \{\langle a, b, c\rangle, \langle b, a, c\rangle, \langle a, b, d, e\rangle, \langle b, a, d, e\rangle, \langle a, b, e, d\rangle, \langle b, a, e, d\rangle, \langle a, d, b, e\rangle, \langle b, e, a, d\rangle\}$.

Note that the example net does not have transitions corresponding to the special start ($\triangleright$) and end ($\square$) activities. Later we will limit ourselves to system nets that have a source place $p_{\triangleright}$, a sink place $p_{\square}$, a start transition $\triangleright$, and an end transition $\square$ such that $p_{\triangleright}$ is the only input place of $\triangleright$ and $p_{\square}$ is the only output place of $\square$. Assuming that the net is connected and $M_{init} = [p_{\triangleright}]$ and $M_{final} = [p_{\square}]$ this implies that all accepting traces start with $\triangleright$ and end with $\square$. However, from a semantical point of view, $\triangleright$ and $\square$ are treated like any other activity.

4. Causal Graphs: Motivation, Discovery, Precision, and Recall

Let us fix a set A of activities, the semantics of formal process models such as Petri nets or BPMN models are provided by the set of possible traces over A they accept/generate. For a given system net $SN = (N, M_{init}, M_{final})$ the set of traces $behav(SN)$ it accepts is a subset of A^*. When evaluating a process discovery approach on a discovered model SN, the event log $L \in \mathcal{B}(A^*)$ used as input is compared with $behav(SN)$. Hence, process models are viewed as binary classifiers stating whether a trace $\sigma \in A^*$ is possible ($\sigma \in behav(SN)$) or not ($\sigma \notin behav(SN)$). In this paper, we would like to move beyond this narrow view and consider both formal and semi-formal constructs by incorporating information that cannot be mapped onto a Petri-net place or BPMN gateway. As mentioned, there may not be a corresponding construct due to the representational bias of the language or there may be not enough evidence in the event log to justify adding a place or gateway. Therefore, we need to abandon the idea that the semantics of a model are fully defined by the set of possible traces. We propose *causal graphs* as an additional way to characterize behaviors, next to sets of traces. We use such causal graphs during the discovery of hybrid process models and to evaluate process discovery results (next to the classical measures that consider process models as binary classifiers). In the next sections, we define causal graphs as well as new metrics for their evaluation.

4.1. Causal Graphs

A *formal process model* is able to make firm statements about the inclusion or exclusion of traces, e.g., trace $\langle \triangleright, a, b, c, d, \square \rangle$ fits the model or not. *Informal process models* are unable to make such precise statements about traces. Event logs only show example behavior: (1) logs are typically incomplete (e.g., the data only shows a fraction of all possible interleavings, combinations of choices, or unfoldings) and (2) logs may contain infrequent exceptional behavior where the model should abstract from. Therefore, it is impossible to make conclusive decisions based on event logs. More observations may lead to a higher certainty and the desire to make a formal statement (e.g., "after a there is a choice between b and c"). However, fewer observations and complex dependencies create the desire to remain "vague". Models (a), (b) and (c) in Figure 6 have formal Petri net semantics as described in Definition 5 (the initial and final markings are defined but not indicated explicitly: the source places are initially marked and the sink places are the only places marked in the final markings). Models (a) and (b) in

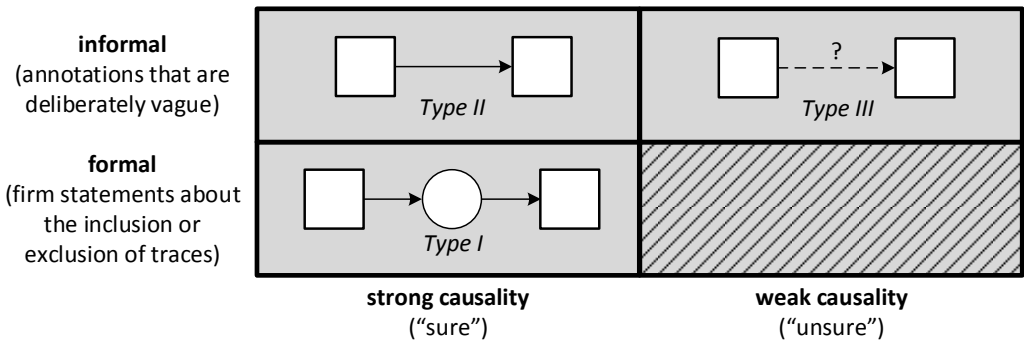


Figure 9: The strength of a causality and the formality of a modeling construct are orthogonal. However, it makes less sense to express a weak causality in a formal manner.

Figure 5 are informal and, therefore, unable to classify traces into fitting and non-fitting.

In essence, process models describe *causalities* between activities. Depending on the evidence in the data we can distinguish between stronger ("sure") or weaker ("unsure") causalities.[2] The strength of a causal relation expresses the level of confidence. A strong causality between two activities a and b suggests that one is quite sure that activity a causes activity b to happen later in time. This does not mean that a is always followed by b. The occurrence of b may depend on other factors, e.g., b requires c to happen concurrently or a only increases the likelihood of b.

The strength of a causality and the formality of a modeling construct are orthogonal dimensions, as shown in Figure 9. Indeed, both strong and weak causalities can be expressed formally and informally. Even when one is not sure, one can still use a formally specified modeling construct. Moreover, both notions may be local, e.g., parts of the process model are more certain or modeled precisely whereas other parts are less clear and therefore kept vague.

As Figure 9 suggests, it seems undesirable to express a weak causality using a formal construct. Moreover, depending on the *representational bias* of the modeling notation, strong causalities may not be expressed easily. The modeling notation may not support concurrency, duplicate activities, unstructured models, long-term dependencies, OR-joins, etc. Attempts to express behavior incompati-

[2]We use two classes two characterize the causal relation strength: "sure" and "unsure". This is a design choice. Choosing more levels of causality strength would have required more thresholds and make the model and approach more complex.

ble with the representational bias of the modeling notation in a formal model are doomed to fail. For instance the α-algorithm and the ILP miner can only discover Petri nets where all transitions have a unique and visible label. The inductive miner can only discover block-structured models where all activities are unique. This results in models overfitting or underfitting. Hence, *things that cannot be expressed easily in an exact manner can only be captured in an informal way, e.g., by using annotations that are deliberately vague and non-executable.* Instead, we aim to combine the best of both worlds by considering *causalities* next to the formal language of a model in terms of the traces it can produce. The first step to achieve this result is introducing *causal graphs*.

A causal graph, which is a variant of the one discovered by the heuristic miner [1, 7], is a directed graph with activities as nodes. There is always a unique start activity ($\triangleright$) and end activity ($\square$) and there are the two kinds of *strong* and *weak* causal relations corresponding to the two columns in Figure 9.

Definition 6 (Causal Graph). *A* causal graph *is a tuple* $G = (A, R_S, R_W)$ *where* A *is the set of activities including start and end activities (i.e.,* $\{\triangleright, \square\} \subseteq A$*),* $R_S \subseteq A \times A$ *is the set of strong causal relations,* $R_W \subseteq A \times A$ *is the set of weak causal relations, and* $R_S \cap R_W = \emptyset$ *(relations are disjoint).* $\mathcal{U}_G$ *is the set of all causal graphs.*

Figure 10 shows a causal graph derived from the event log also used to discover the models in Figure 5 and Figure 6. Analogously to the hybrid model example in Figure 7, the dashed arcs with question marks represent two weak causal relations, while the others represent strong causal relations.

Note that the causal graph in Figure 10 does *not* have formal semantics in the classical sense, i.e., a causal graph is *not* a binary trace classifier and should be viewed as semi-formal.

4.2. Discovering Causal Graphs

Although causal graphs are semi-formal, we want to discover such graphs from event data and use these to discover and evaluate process models. We assume that a *causal graph discovery function* provides the strong and weak causal relations. Note that this is a necessary *assumption* to discuss hybrid process models in a meaningful manner.

Definition 7 (Causal Graph Discovery). *A causal graph discovery function is a function* $disc_{cg} \in \mathcal{U}_L \rightarrow \mathcal{U}_G$ *that constructs a causal graph* $disc_{cg}(L) = (A, R_S, R_W)$ *for any event log* $L \in \mathcal{U}_L$ *over* A*.*

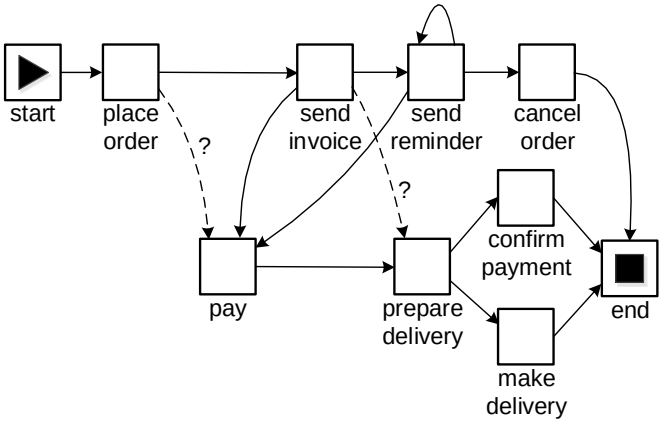


Figure 10: A causal graph: nodes correspond to activities and arcs correspond to causal relations.

There are many possible algorithms to construct a causal graph from a log. Here we use a variant of the algorithm used by the heuristic miner [1, 7] tailored to hybrid discovery (i.e., different types of arcs) while aiming for parameters that are intuitive and can be used interactively (e.g., thresholds can be changed seamlessly while instantly showing the resulting graph). Note that we clearly separate the identification of causalities from the discovery of formal process modeling constructs such as places or gateways. Some preliminary definitions are required for the causal graph discovery algorithm and they are presented next.

Definition 8 (Log-Based Properties). *Let A be a set of activities, $L \in \mathcal{U}_L$ be an event log over A and $\{a, b\} \subseteq A$.*

- $\#(a, L) = \sum_{\sigma \in L} |\{i \in dom(\sigma) \mid \sigma(i) = a\}|$ *counts the number of a's in event log L.*
- $\#(X, L) = \sum_{x \in X} \#(x, L)$ *counts the number of $X \subseteq A$ activities in L.*
- $\#(a, b, L) = \sum_{\sigma \in L} |\{i \in dom(\sigma) \setminus \{|\sigma|\} \mid \sigma(i) = a \ \wedge \ \sigma(i+1) = b\}|$ *counts the number of times a is directly followed by b in event log L.*
- $\#(*, b, L) = \sum_{\sigma \in L} |\{i \in dom(\sigma) \setminus \{|\sigma|\} \mid \sigma(i+1) = b\}|$ *counts the number of times b is preceded by some activity.*
- $\#(a, *, L) = \sum_{\sigma \in L} |\{i \in dom(\sigma) \setminus \{|\sigma|\} \mid \sigma(i) = a\}|$ *counts the number of times a is succeeded by some activity.*

- $Rel1(a,b,L) = \dfrac{2\#(a,b,L)}{\#(a,*,L)+\#(*,b,L)}$ *counts the strength of relation* (a, b) *relative to the split and join behavior of activities* a *and* b*.*

- $Rel2_c(a,b,L) = \begin{cases} \frac{\#(a,b,L)-\#(b,a,L)}{\#(a,b,L)+\#(b,a,L)+c} & \text{if } \#(a,b,L)-\#(b,a,L) > 0 \\ \frac{\#(a,b,L)}{\#(a,b,L)+c} & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$
counts the strength of relation (a, b) *taking into account concurrency and loops using parameter* $c \in \mathbb{R}^+$ *(default* $c = 1$*).*

- $Caus_{c,w}(a,b,L) = w \cdot Rel1(a,b,L) + (1-w) \cdot Rel2_c(a,b,L)$ *takes the weighted average of both relations where* $w \in [0,1]$ *is a parameter indicating the relative importance of the first relation. If* $w = 1$*, we only use* $Rel1(a,b,L)$*. If* $w = 0$*, we only use* $Rel2_c(a,b,L)$*. If* $w = 0.5$*, then both are of equal weight.*

$Rel1(a,b,L)$, $Rel2_c(a,b,L)$, and $Caus_{c,w}(a,b,L)$ all produce values between 0 (no relation) and 1 (perfect relation), so that higher values correspond to a stronger causality relationship. Using the properties in Definition 8, we can now define a concrete function $disc_{cg}$ to create causal graphs. All activities that occur in at least t_{freq} traces in the event log are included as nodes. The strength of relations between remaining activities (based on $Caus_{c,w}$) are used to infer causal relations. Note that, differently from activity frequency, the strength of a causal relation between two activities measures how sure we are on whether a causal relation between the activities exists. t_{R_S} and t_{R_W} are thresholds for strong respectively weak causal relations. Parameter w determines the relative importance of $Rel1$ and $Rel2_c$. Parameter c is typically set to 1.

Note that $Rel2_c(a,b,L)$ is similar to one of the features also used in the heuristic miner [7]. However, here negative values are mapped to 0 to get a value between 0 and 1. Again we would like to stress that other measures could be used to measure the strength of the relations among activities and that the contribution of this work is not in the generation of causal graphs, but in the construction of hybrid process models (although the quality of the discovered hybrid process model could depend on the quality of the causal graph).

Definition 9 (Concrete Causal Graph Discovery Technique). *Let* A *be a set of activities,* $L \in \mathcal{U}_L$ *be an event log over* A *and let* $t_{freq} \in \mathbb{N}^+$*,* $c \in \mathbb{R}^+$*,* $w \in [0,1]$*,* $t_{R_S} \in [0,1]$*,* $t_{R_W} \in [0,1]$ *be parameters such that* $t_{R_S} \geq t_{R_W}$*. The corresponding causal graph is* $G = disc_{cg}(L) = (A', R_S, R_W)$ *where*

- $A' = \{a \in A \mid \#(a, L) \geq t_{freq}\} \cup \{\triangleright, \square\}$ *is the set of activities that meet the threshold (the start and end activities are always included).*
- $R_S = \{(a, b) \in A' \times A' \mid Caus_{c,w}(a, b, L\restriction_{A'}) \geq t_{R_S}\}$ *is the set of strong causal relations.*
- $R_W = \{(a, b) \in A' \times A' \mid t_{R_S} > Caus_{c,w}(a, b, L\restriction_{A'}) \geq t_{R_W}\}$ *is the set of weak causal relations.*

Figure 10 shows a causal graph constructed using parameters $t_{freq} = 1000$, $c = 1$, $w = 0.2$, $t_{R_S} = 0.8$, and $t_{R_W} = 0.75$.

In the remainder, we assume $disc_{cg}(L)$ to be the causal graph describing the event log in the best way. This is an *assumption* parameterized by t_{R_S} and t_{R_W}, but it is a necessary one when it comes to evaluate (hybrid) process models (it is possible to pick a different causal graph discovery function $disc_{cg} \in \mathcal{U}_L \rightarrow \mathcal{U}_G$).

4.3. Causal-Graph-Based Precision and Recall

Process discovery techniques can be evaluated using a range of indicators referring to *recall* (ability to replay the observed behavior, often called fitness), *precision* (avoiding underfitting, i.e., penalizing behavior possible according to the model but never observed), *generalization* (avoiding overfitting, i.e., the ability to anticipate behavior not observed yet), and *simplicity* (is the model easy to understand) [1]. These notions are also relevant for hybrid process models, but they ignore the two dimensions shown in Figure 9. Indeed classical process models only consider the *Type I* relation but do not consider the top row in Figure 9 with *Type II* and *Type III* relations. Therefore, in order to evaluate and compare our approach, we need to: *(i)* show to create a causal graph for a formal model and *(ii)* define causal-graph-based precision and recall. For systems nets (i.e., Petri nets with an initial and final marking), we define the corresponding causal graph as follows. The Petri net transitions define the nodes (i.e., activities) of the causal graph. The strong causal relations are all pairs of transitions (i.e., activities) connected through a place. There are no weak causal relations, because there are no "unsure" relationships.

Definition 10 (Causal Graph of a System Net). *Let $SN = (N, M_{init}, M_{final})$ be a system net. $G(SN) = (A, R_S, R_W)$ is the causal graph of SN where $A = T$, $R_S = \{(a_1, a_2) \in A \times A \mid a_1 \bullet \cap \bullet a_2 \neq \emptyset\}$, and $R_W = \emptyset$.*

As mentioned before, we will focus on system nets that have a source place $p_{\triangleright}$, a sink place $p_{\square}$, a start transition $\triangleright$, and an end transition $\square$, and event logs where each trace starts with $\triangleright$ and ends with $\square$. Hence, causal graphs always include a start activity $\triangleright$ and an end activity $\square$.

The same approach can be used when using other graphical models like BPMN models or UML activity diagrams. Based on a BPMN model we can derive the strong causal relations by considering the connections through gateways, just like we considered connections through shared places in Definition 10. Considering traditional process model notations not allowing for "unsure" relationships, there will be no weak causal relations. However, the hybrid process models introduced later will have such relationships.

Now that we have a causal graph for event logs and process models, we can define *causal-graph-based precision and recall*.

Definition 11 (Causal-Graph-Based Precision and Recall). *Let $G^L = (A^L, R_S^L, R_W^L)$ be a causal graph for event log L and $G^M = (A^M, R_S^M, R_W^M)$ be a causal graph for process model M.*

- *Strong Causal Recall: $SCR(L, M) = \frac{\left|R_S^L \cap R_S^M\right|}{\left|R_S^L\right|}$ is the fraction of strong causalities according to the event log supported by the process model.*
- *Strong Causal Precision: $SCP(L, M) = \frac{\left|R_S^M \cap R_S^L\right|}{\left|R_S^M\right|}$ is the fraction of strong causalities according to the process model supported by the event log.*
- *Weak Causal Recall: $WCR(L, M) = \frac{\left|R_W^L \cap R_W^M\right|}{\left|R_W^L\right|}$ is the fraction of weak causalities according to the event log supported by the process model.*
- *Weak Causal Precision: $WCP(L, M) = \frac{\left|R_W^M \cap R_W^L\right|}{\left|R_W^M\right|}$ is the fraction of weak causalities according to the process model supported by the event log.*

Note that $SCR(L, M)$, $SCP(L, M)$, $WCR(L, M)$, and $WCP(L, M)$ are defined independent of the process model notation used. Given a system net $M = SN = (N, M_{init}, M_{final})$, we can compute $G^M = (A^M, R_S^M, R_W^M) = G(SN)$ as shown in Definition 10. Given an event log $L \in \mathcal{B}(A^*)$, we can compute $G^L = (A^L, R_S^L, R_W^L) = disc_{cg}(L)$ as shown in Definition 9 (parameterized by t_{R_S} and t_{R_W}).

For traditional process model notations not allowing for "unsure" relationships, we have $R_W^M = \emptyset$ by definition (cf. Definition 10). This implies that $WCR(L, M) = 0$ and $WCP(L, M) = 1$ (assuming that $0/0 = 1$).

In the evaluation of our discovery approach, we use causal-graph-based recall and precision, next to the more traditional notions of recall and precision.

5. Hybrid Petri Nets

The aim of this paper is to incorporate in our hybrid process model information that cannot be mapped onto a Petri-net place or BPMN gateway, e.g., due to representational bias issues or to the lack of evidence in the log to justify the addition of these elements. We therefore introduce a new type of hybrid process models that contain both formal Petri net places and informal causal graph sure and unsure arcs, so as to have the three types of connections shown in Figure 9. The resulting model is *hybrid Petri nets*, as we augment traditional Petri nets with such arcs. However, we remark that our ideas are *generic* and could also apply to other notations such as BPMN, UML activity diagrams, etc.

Figure 7 shows an example of a hybrid Petri net discovered based on the event log in Figure 4, which is the same log used to create the models in Figure 5 and Figure 6. Strong causalities are expressed through conventional places and arcs as well as sure arcs (which directly connect transitions), while weak causalities are represented using unsure arcs. The figure also uses special symbols for the start and end activities ($\triangleright$ and $\square$) as introduced in Definition 1, but the semantics of HSN, defined below, do not depend on this.

Definition 12 (Hybrid Petri Net). *A* hybrid Petri net *is a tuple* $HPN = (P, T, F_I, F_{II}, F_{III})$ *where* (P, T, F_I) *is a Petri net,* $F_{II} \subseteq T \times T$*, and* $F_{III} \subseteq T \times T$ *such that* $\widehat{F_I}$*,* F_{II}*, and* F_{III} *are pairwise disjoint. Arcs of Type I (*$(p, t) \in F_I$ *or* $(t, p) \in F_I$*) are the normal arcs connecting a place to a transition or vice versa. Arcs of Type II (*$(t_1, t_2) \in F_{II}$*) are arcs indicating a strong causality between two transitions (sure arcs). Arcs of Type III (*$(t_1, t_2) \in F_{III}$*) are arcs indicating a weak causality between two transitions (unsure arcs).*

In order to reason about the set of accepted traces, we introduce again initial and final markings and define the notion of a *hybrid system net*, which essentially states that transitions, places, and normal (*Type I*) arcs have the formal semantics defined in Section 3.3, while the other two types of arcs are informal and do not include or exclude traces.

Definition 13 (Hybrid System Net). *A* hybrid system net *is a triplet* $HSN = (HPN, M_{init}, M_{final})$ *where* $HPN = (P, T, F_I, F_{II}, F_{III})$ *is a hybrid Petri net,* $M_{init} \in \mathcal{B}(P)$ *is the initial marking, and* $M_{final} \in \mathcal{B}(P)$ *is the final marking.* $\mathcal{U}_{HSN}$ *is the set of all possible hybrid system nets.* $behav(HSN)$ *is defined as in Definition 5 while ignoring the sure and unsure arcs (i.e., remove* F_{II} *and* F_{III}*).*

Recall that a Petri net without any places allows for any behavior and adding a place can only restrict behavior. In this light, a sure arc $(t_1, t_2) \in F_{II}$ should be interpreted as a strong causal relationship that cannot be expressed (easily) in terms of a place connecting t_1 and t_2. An unsure arc $(t_1, t_2) \in F_{III}$ is a suspected causal relationship that is too weak to justify a place connecting t_1 and t_2.

Next, we define the causal graph of a hybrid Petri net. In this way, causal-graph-based precision and recall (as presented in Definition 11) can be used to directly evaluate and compare our models with others, along with more traditional evaluation measures which ignore the "unsure" or "inexpressible" causalities.

Definition 14 (Causal-Graph of a hybrid Petri net.). *Let* HSN *be a hybrid system net with* $HSN = (HPN, M_{init}, M_{final})$ *and* $HPN = (P, T, F_I, F_{II}, F_{III})$. $G(HSN) = (A, R_S, R_W)$ *is the causal graph of* HSN *where* $A = T$, $R_S = \{(a_1, a_2) \in A \times A \mid a_1 \bullet \cap \bullet a_2 \neq \emptyset\} \cup F_{II}$, *and* $R_W = F_{III}$.

Assuming $G^L = (A^L, R_S^L, R_W^L)$ be a causal graph for some event log L and $G^M = (A^M, R_S^M, R_W^M) = G(HSN)$ for a hybrid Petri net $M = HSN$, the casual-graph-based precision and recall are defined as before: $SCR(L, M) = \left|R_S^L \cap R_S^M\right| / \left|R_S^L\right|$ (strong causal recall), $SCP(L, M) = \left|R_S^M \cap R_S^L\right| / \left|R_S^M\right|$ (strong causal precision), $WCR(L, M) = \left|R_W^L \cap R_W^M\right| / \left|R_W^L\right|$ (weak causal recall), and $WCP(L, M) = \left|R_W^M \cap R_W^L\right| / \left|R_W^M\right|$ (weak causal precision).

Section 8 makes use of these and other metrics to perform an extensive evaluation of our approach.

6. Discovering Hybrid Process Models

Our hybrid process discovery technique uses a *two-step approach*. In the first step, we discover a *causal graph* with strong and weak causal relations (as presented in Section 4). In the second step, we build the actual hybrid Petri net as follows. First, we take as input the causal graph and we aim at converting as many strong causal relations as possible into Petri net places. This is done by generating candidate places and then filtering them based on how well –according to

threshold parameters set by the user– they replay log traces. After that, the strong causalities (in the causal graph) that have not been expressed in term of places are added to the hybrid model as sure arcs, and unsure arcs (again, in the causal graph) are added to the model for weak causal relations.

The procedure of identifying and evaluating the quality of candidate places covers a major role in the discovery technique and it requires reasoning about places in the context of event data, which is introduced next. Section 6.2 makes use of them to describe the actual discovery algorithm and finally Section 6.3 describes enhancement techniques to improve the performances.

6.1. Evaluating the Quality of Places in Relation to the Event Log

We begin by recalling that a (traditional) Petri net without places allows for any behavior, i.e., if $SN = ((\emptyset, A, \emptyset), [], [])$, then $behav(SN) = A^*$, i.e., all possible traces involving activities in A. Therefore process discovery can be viewed as the process of adding places to constrain the model to the behavior seen in the event log. Let us then elaborate on the role of places in this setting.

6.1.1. Place Properties

Events do not refer to place names, but only to activities represented by transitions. Fortunately, Petri-net places can be fully characterized in terms of the transitions they connect. Here, we assume that no two places have identical input and output transitions, because if two places have identical input and output transitions, at least one of them is redundant.

Definition 15 (Places). *Let A be a set of activities. The set of possible places is denoted by $\mathcal{P}_A = \{(I, O) \mid I \subseteq A \wedge O \subseteq A\}$. Place $p = (I, O) \in \mathcal{P}_A$ has $\bullet p = I$ as input transitions and $p\bullet = O$ as output transitions. $\mathcal{P}_A^+ = \{(I, O) \in \mathcal{P}_A \mid I \neq \emptyset \wedge O \neq \emptyset\}$ is the set of proper places, i.e., places with at least one input transition and one output transition.*

For example the Petri net in Figure 8 has four places: $p1 = (\emptyset, \{a\})$, $p2 = (\emptyset, \{b\})$, $p3 = (\{a\}, \{c, d\})$, and $p4 = (\{b\}, \{c, e\})$.

We will use the terms transition and activity interchangeably, because we assume a one-to-one mapping between transitions and activities, i.e., $T = A$. To compare sets of activities, we used the following two shorthands: (1) $A_1 \parallel A_2$ denotes that $A_1 \cap A_2 = \emptyset$ and (2) $A_1 \div A_2$ denotes that $A_1 \cap A_2 \neq \emptyset$, $A_1 \setminus A_2 \neq \emptyset$, and $A_2 \setminus A_1 \neq \emptyset$. These shorthands are used to characterize the relationship between two sets of activities. For any pair of two non-empty sets of activities A_1

and A_2, *precisely* one of the following statements holds: $A_1 = A_2$ (equality), $A_1 \parallel A_2$ (non-overlapping), $A_1 \subset A_2$ (proper subset), $A_1 \supset A_2$ (proper superset), or $A_1 \div A_2$ (incomparable). Similarly, we can relate places. To do this, we first introduce some notations.

Definition 16 (Place Notations). *Let $p_1 = (I_1, O_1) \in \mathcal{P}_A$ and $p_2 = (I_2, O_2) \in \mathcal{P}_A$ be two places. These places can be combined to form new places:*

- $p_1 \sqcap p_2 = (I_1 \cap I_2, O_1 \cap O_2) \in \mathcal{P}_A$,
- $p_1 \sqcup p_2 = (I_1 \cup I_2, O_1 \cup O_2) \in \mathcal{P}_A$, *and*
- $p_1 \otimes p_2 = ((I_1 \cup I_2) \setminus (I_1 \cap I_2), (O_1 \cup O_2) \setminus (O_1 \cap O_2)) \in \mathcal{P}_A$.

We can relate p_1 and p_2 in different ways:

- $p_1 = p_2$ *if and only if* $I_1 = I_2$ *and* $O_1 = O_2$ *(equality),*
- $p_1 \parallel p_2$ *if and only if* $p_1 \sqcap p_2 = (\emptyset, \emptyset)$ *(non-overlapping),*
- $p_1 \sqsubset p_2$ *if and only if* $I_1 \subseteq I_2$, $O_1 \subseteq O_2$, *and* $p_1 \neq p_2$ *(proper subset), and*
- $p_1 \div p_2$ *if and only if* $p_1 \neq p_2$, $p_1 \nparallel p_2$, $p_1 \not\sqsubset p_2$ *and* $p_1 \not\sqsupset p_2$ *(incomparable).*

Any two proper places $p_1, p_2 \in \mathcal{P}_A^+$ are related in precisely one of the following ways: $p_1 = p_2$ (equality), $p_1 \parallel p_2$ (non-overlapping), $p_1 \sqsubset p_2$ (proper subset), $p_1 \sqsupset p_2$ (proper superset), or $p_1 \div p_2$ (incomparable). For the Petri net in Figure 8: $p1 \parallel p2$, $p1 \parallel p3$, and $p3 \div p4$.

6.1.2. Scoring Places.

To evaluate the quality of a place, we need to check which traces in an event log allow for it. When replaying a trace, one should only consume tokens that are actually there and at the end of the trace the place should be empty. $behav(SN)$ was defined for the system net as a whole. However, one can also check this for each place separately. This is what we will later do in the discovery approach.

Definition 17 (Replayable Trace). *Let $p = (I, O) \in \mathcal{P}_A$ be a place and A a set of activities. A trace $\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in A^*$ is perfectly replayable with respect to place p if and only if*

- *for all $k \in \{1, 2, \ldots, n\}$: $|\{1 \leq i < k \mid a_i \in I\}| \geq |\{1 \leq i \leq k \mid a_i \in O\}|$ (place p cannot "go negative" while replaying the trace) and*

- $|\{1 \leq i \leq n \mid a_i \in I\}| = |\{1 \leq i \leq n \mid a_i \in O\}|$ *(place p is empty at end).*

We write $\checkmark(p, \sigma)$ if σ is perfectly replayable with respect to place p. $act(p, \sigma) = \exists_{a \in \sigma}\ a \in (I \cup O)$ denotes whether the place has been activated, i.e., a token was consumed or produced for it in σ.

The expression $\not\checkmark(p, \sigma)$ denotes that σ is *not* perfectly replayable with respect to place p. Consider the system net in Figure 8 and the three traces: $\sigma_1 = \langle a, b, c\rangle$, $\sigma_2 = \langle a, e, b, d\rangle$, and $\sigma_3 = \langle a, b, d\rangle$. σ_1 is perfectly replayable for all four places: $\checkmark(p1, \sigma_1)$, $\checkmark(p2, \sigma_1)$, etc. σ_2 is perfectly replayable for three places (i.e., $\checkmark(p1, \sigma_2)$, $\checkmark(p2, \sigma_2)$, $\checkmark(p3, \sigma_2)$), but not for $p4$: $\not\checkmark(p4, \sigma_2)$ because e requires b to happen first. σ_3 is perfectly replayable for the same three places but again not for $p4$: $\not\checkmark(p4, \sigma_3)$ because a token remains in $p4$.

Note that $\checkmark(p, \sigma)$ if σ is a trace of the system net having only one place $p = (I, O)$. Formally, $\checkmark(p, \sigma)$ if and only if $\sigma \in behav(SN_p)$ with $SN_p = ((P_p, T_p, F_p), [\,], [\,])$, $P_p = \{p\}$, $T_p = A$, $F_p = \{(t, p) \mid t \in I\} \cup \{(p, t) \mid t \in O\}$.

For any pair of places p_1 and p_2 and trace $\sigma \in A^*$: $act(p_1 \otimes p_2, \sigma)$ if σ contains at least one activity that either activates p_1 or p_2 but not both of them. If $\neg act(p_1 \otimes p_2, \sigma)$, then for any executed activity $a \in \sigma$: *(i)* $a \in {\bullet p_1}$ if and only if $a \in {\bullet p_2}$ and *(ii)* $a \in p_1{\bullet}$ if and only if $a \in p_2{\bullet}$. Therefore, while replaying σ, at any point in time, places p_1 and p_2 have the same number of tokens if $\neg act(p_1 \otimes p_2, \sigma)$.

To evaluate candidate places generated by our discovery algorithm, one can use different quality scores. Here, we consider two obvious choices, which both provide an idea of how good the traces fit the place, thus allowing us to get models of high fitness.

Definition 18 (Place Scores). *Let $L \in \mathcal{U}_L$ be an event log. For any place $p = (I, O) \in \mathcal{P}_A$, we define the following scores:*

- $score_{freq}(p, L) = \frac{|\,[\sigma \in L \mid \checkmark(p,\sigma)]\,|}{|L|}$ *is the fraction of fitting traces for p, and*
- $score_{rel}(p, L) = \frac{|\,[\sigma \in L \mid \checkmark(p,\sigma) \wedge act(p,\sigma)]\,|}{|\,[\sigma \in L \mid act(p,\sigma)]\,|}$ *is the fraction of fitting traces for p that have been activated.*

Consider the system net in Figure 8 and the event log $L = [\langle a, b, c\rangle^8, \langle a, e, b, d\rangle^1, \langle a, b, d\rangle^1]$: $score_{freq}(p1, L) = score_{freq}(p2, L) = score_{freq}(p3, L) = 1$. Moreover, $score_{freq}(p4, L) = 0.8$. In this particular example, we find the same scores for $score_{rel}$. In general this is not the case. Take $L = [\langle\,\rangle^8, \langle c\rangle^2]$, now $score_{freq}(p3, L) = score_{freq}(p4, L) = 0.8$ and $score_{rel}(p3, L) = score_{rel}(p4, L) = 0$. Later, we will argue in favor of $score_{rel}$.

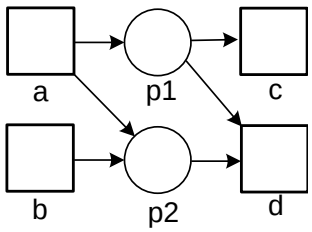


Figure 11: Petri net used to explain the notion of conflict: $p1\#p2$ because $p1 \prec p2$.

6.1.3. Conflicting and Redundant Places.

Our discovery approach aims to add places to formally constrain behavior. Often there are many candidate places. To limit the search space we do not want to add places that are conflicting, according to the definition below.

Definition 19 (Conflict). *The places $p_1 = (I_1, O_1) \in \mathcal{P}_A^+$ and $p_2 = (I_2, O_2) \in \mathcal{P}_A^+$ are conflicting (notation $p_1 \# p_2$) if one of the following two cases holds:*

- *$I_1 \subseteq I_2$, $O_2 \subseteq O_1$, and $p_1 \neq p_2$ (notation $p_1 \prec p_2$) or*
- *$I_2 \subseteq I_1$, $O_1 \subseteq O_2$, and $p_1 \neq p_2$ (notation $p_1 \succ p_2$).*

If $p_1 \prec p_2$, then p_1 cannot have more tokens than p_2 at any point in time when replaying a trace from some event log. Similarly, if $p_1 \succ p_2$, then $p1$ cannot have less tokens than p_2. In Figure 11, $p1\#p2$ because $p1 \prec p2$.

Lemma 1. *Let $p_1, p_2 \in \mathcal{P}_A^+$ be two proper places and $\sigma \in A^*$ a trace. If $\checkmark(p_1, \sigma)$, $p_1 \# p_2$, and $act(p_1 \otimes p_2, \sigma)$, then $\not\checkmark(p_2, \sigma)$.*

Proof 1. *Assume $\checkmark(p_1, \sigma)$, $p_1 \# p_2$, and $act(p_1 \otimes p_2, \sigma)$. Hence, $p_1 \prec p_2$ or $p_1 \succ p_2$. If $p_1 \prec p_2$, then the number of tokens in p_2 is always at least the number of tokens in p_1 for any sequence, including σ. In fact, in σ there is at least one additional token produced or a token was not consumed (because $act(p_1 \otimes p_2, \sigma)$). Because p_1 ends empty, p_2 must have a remaining token at the end. Hence, σ cannot be fitting. If $p_1 \succ p_2$, then the number of tokens in p_2 is always at most the number of tokens in p_1. In fact there is at least one additional token consumed or a token was not produced (because $act(p_1 \otimes p_2, \sigma)$). Because p_1 ends empty, p_2 must have a missing token at the end. Hence, σ cannot be fitting.* □

Consider the places in Figure 11. Note that $p1 \otimes p2 = (\{b\}, \{c\})$. Since $\checkmark(p1, \langle a, c\rangle)$, $p1 \# p2$, and $act(p1 \otimes p2, \langle a, c\rangle)$, Lemma 1 tells us that $\langle a, c\rangle$ is not perfectly replayable with respect to $p2$, i.e., $\not\checkmark(p2, \langle a, c\rangle)$ (indeed a token remains in $p2$). Since $\checkmark(p2, \langle b, d\rangle)$, $p1 \# p2$, and $act(p1 \otimes p2, \langle b, d\rangle)$, Lemma 1 tells us that $\not\checkmark(p_1, \langle b, d\rangle)$ (indeed a token is missing in $p1$).

During discovery, we would like to avoid adding conflicting places. We will use an iterative approach adding places one by one. Therefore, we would like to check conflicts with respect to combinations of existing places.

Definition 20 (Conflict with a set of places). *Proper place $p \in \mathcal{P}_A^+$ and the set of proper places $P \subseteq \mathcal{P}_A^+$ (with $p \notin P$) are conflicting (notation $p \# P$) if there is a non-empty subset $P' = \{p_1, p_2, \ldots, p_n\} \subseteq P$ such that $p_i \parallel p_j$ for any $1 \leq i < j \leq n$ and $p \# (p_1 \sqcup p_2 \sqcup \ldots \sqcup p_n)$.*

The following lemma shows why we would like to avoid adding conflicting places. We will use this to avoid spending time on checking the quality of such places.

Lemma 2. *Let $p \in \mathcal{P}_A^+$ be a proper place and $P \subseteq \mathcal{P}_A^+$ a set of proper places such that $p \# P$. By definition there is a non-empty subset $P' = \{p_1, p_2, \ldots, p_n\} \subseteq P$ such that $p_i \parallel p_j$ for any $1 \leq i < j \leq n$ and $p \# p'$ with $p' = p_1 \sqcup p_2 \sqcup \ldots \sqcup p_n$.*

For any trace $\sigma \in A^*$: if $\checkmark(p_i, \sigma)$ for all $1 \leq i \leq n$ and $act(p \otimes p', \sigma)$, then $\not\checkmark(p, \sigma)$.

Proof 2. *The place p' is a proper place ($p' \in \mathcal{P}_A^+$) because $\bullet p' \neq \emptyset$ and $p' \bullet \neq \emptyset$. Assume all p_i places are fitting σ. Hence, p' is fitting and the number of tokens in p' is equal to the sum of the tokens in the places in P'. Hence, $\checkmark(p', \sigma)$. By applying Lemma 1 to places p and p', we have that the requirement $act(p \otimes p', \sigma)$ ensures that p cannot be fitting when p' is already fitting.* □

If $p_1 \parallel p_2$, then the number of tokens in $p_3 = p_1 \sqcup p_2$ is identical to the total number of tokens in both p_1 and p_2. This means the traces that are perfectly replayable with respect to places p_1 and p_2, are also perfectly replayable with respect to places p_3.

Lemma 3. *Let $p_1, p_2 \in \mathcal{P}_A^+$ be two proper places and $\sigma \in A^*$ a trace. If $\checkmark(p_1, \sigma)$, $\checkmark(p_2, \sigma)$, and $p_1 \parallel p_2$, then $\checkmark(p_1 \sqcup p_2, \sigma)$.*

Proof 3. *Assume* $\checkmark(p_1, \sigma)$, $\checkmark(p_2, \sigma)$, $p_1 \parallel p_2$, *and* $p_3 = p_1 \sqcup p_2$. *The number of tokens in* p_3 *is always the sum of the number of tokens in* p_1 *and* p_2. *Therefore, it cannot turn negative and equals zero at the end. Hence,* σ *is perfectly fitting* p_3. □

Definition 21 (Redundant). *Proper place* $p \in \mathcal{P}_A^+$ *is redundant with respect to a set of proper places* $P \subseteq \mathcal{P}_A^+$ *(notation* $p \sim P$*) if there is a non-empty subset* $P' = \{p_1, p_2, \ldots, p_n\} \subseteq P$ *such that* $p_i \parallel p_j$ *for any* $1 \leq i < j \leq n$ *and* $p = (p_1 \sqcup p_2 \sqcup \ldots \sqcup p_n)$.

We would like to avoid adding redundant places to the discovered model, because they no further constrain the behavior and only complicate the model.

6.2. Hybrid System Net Discovery Approach

In this section we describe the approach for the discovery of the hybrid system net, but first we provide the signature of the function we want to create.

Definition 22 (Hybrid System Net Discovery). *A hybrid system net discovery function* $disc_{hsn} \in (\mathcal{U}_L \times \mathcal{U}_G) \to \mathcal{U}_{HSN}$ *is a function that for any event log* L *and causal graph* G *discovers a hybrid system net* $disc_{hsn}(L, G) \in \mathcal{U}_{HSN}$.

Just like there are many algorithms possible to create a causal graph, there are also multiple ways to construct a hybrid system net from an event log and causal graph. The minimal consistency requirements can be defined as follows.

Definition 23 (Consistent). *Let* $L \in \mathcal{U}_L$ *be an event log, let* $G = (A, R_S, R_W) \in \mathcal{U}_G$ *be a causal graph, and let* $HSN = (HPN, M_{init}, M_{final}) \in \mathcal{U}_{HSN}$ *with* $HPN = (P, T, F_I, F_{II}, F_{III})$ *be a hybrid system net.* L, G, *and* HSN *are* consistent *if and only if:*

(c1) $T = A \subseteq \bigcup_{\sigma \in L} \{a \in \sigma\}$ *is the set of activities*

(c2) a source and a sink place exist in HSN $\{p_\triangleright, p_\square\} \subseteq P$, $M_{init} = [p_\triangleright]$ *and* $M_{final} = [p_\square]$ *such that*

(c3) $F_I \cap ((\{p_\triangleright, p_\square\} \times T) \cup (T \times \{p_\triangleright, p_\square\})) = \{(p_\triangleright, \triangleright), (\square, p_\square)\}$,

(c4) for all $p \in P \setminus \{p_\triangleright, p_\square\}$: $\bullet p \neq \emptyset$ *and* $p\bullet \neq \emptyset$,

(c5) $R_S = \widehat{F_I} \cup F_{II}$, $\widehat{F_I} \cap F_{II} = \emptyset$, *and*

(c6) $R_W = F_{III}$.

Intuitively, an event log L, causal graph G, and hybrid system net HSN are consistent if (1) L, G, and HSN refer to the same set of activities all appearing in the event log (but not all activities need to appear in the log) (see $c1$), (2) there is a source place $p_{\triangleright}$ marked in the initial place and enabling start activity $\triangleright$ ($c2$ and $c3$), (3) there is a sink place $p_{\square}$ marked in the final marking and connected to end activity $\square$ ($c2$ and $c3$), (4) all other places connect activities ($c3$ and $c4$), (5) there is a one-to-one correspondence between strong causal relations (R_S) and connections through places ($\widehat{F_I}$) or sure arcs (F_{II}) ($c5$), and (6) there is a one-to-one correspondence between weak causal relations (R_W) and unsure arcs (F_{III}) ($c6$).

Consider two activities $a_1, a_2 \in A$ that are frequent enough to be included in the model. These can be related in three different ways: $(a_1, a_2) \in \widehat{F_I}$ if there is a place connecting a_1 and a_2, $(a_1, a_2) \in F_{II}$ if there is no place connecting a_1 and a_2 but there is a strong causal relation between a_1 and a_2 (represented by a sure arc), $(a_1, a_2) \in F_{III}$ if there is a weak causal relation between a_1 and a_2 (represented by an unsure arc).

Any discovery function $disc_{hsn} \in (\mathcal{U}_L \times \mathcal{U}_G) \to \mathcal{U}_{HSN}$ should ensure consistency. In fact, Definition 23 already provided hints on how to discover a hybrid system net. Assume a place $p = (I, O)$ with a non-empty set of input transitions $\bullet p = I$ and a non-empty set of output transitions $p\bullet = O$ is added. $R_S = \widehat{F_I} \cup F_{II}$ implies that $\widehat{F_I} \subseteq R_S$. Hence, $I \times O \subseteq R_S$, i.e., place $p = (I, O)$ can only connect transitions having strong causal relations. Based on this insight we define the notion of valid places. A place is valid if the corresponding causalities exist in the causal graph.

Definition 24 (Valid places). *Let $G = (A, R_S, R_W) \in \mathcal{U}_G$ be a causal graph. The valid places based on G are: $P_G^V = \{(I, O) \in \mathcal{P}_A^+ \mid I \times O \subseteq R_S\}$.*

A place p' is dominated by another place p if it has fewer arcs, $p' \sqsubset p$. If p is valid, then p' is also valid.

Lemma 4 (Monotonicity). *Let P_G^V be the set of valid places. If $p \in P_G^V$, then all dominated places are also in P_G^V, i.e., $\{p' \in \mathcal{P}_A^+ \mid p' \sqsubset p\} \subseteq P_G^V$.*

Candidate places characterize the potential places that should be considered given a causal graph. Candidate places need to be valid, but it is possible to

impose other constraints. For example, to ensure the simplicity of the model, one could consider only places having few connections.

Definition 25 (Candidate places). *Let $G = (A, R_S, R_W) \in \mathcal{U}_G$ be a causal graph. $P_C \subseteq P_G^V$ is a candidate place set based on G.*

Many choices for P_C are possible. In this paper we propose four sets of candidate places in the context of a causal graph. The first set contains all the valid places, while the others impose some constraints on the number of the place input and/or output connections:

- $P_C^{all} = P_G^V$, i.e., all valid places are candidates.
- $P_C^k = \{(I, O) \in P_G^V \mid |I| + |O| \leq k\}$ with $k \geq 2$ meaning that places can have at most k connections.
- $P_C^{k_i,k_O} = \{(I, O) \in P_G^V \mid |I| \leq k_I \wedge |O| \leq k_O\}$ with $k_I, k_O \geq 1$, i.e., the number of input arcs is bounded to k_I and the number of output arcs is bounded to k_O.
- $P_C^{sj} = \{(I, O) \in P_G^V \mid |I| = 1\} \cup \{(I, O) \in P_G^V \mid |O| = 1\}$ meaning that places can have multiple ingoing arcs or multiple outgoing arcs, but not both at the same time.

Given a set of places, we need to evaluate the quality. Earlier we defined: $score_{freq}(p, L) = \frac{|[\sigma \in L | \checkmark(p,\sigma)]|}{|L|}$ and $score_{rel}(p, L) = \frac{|[\sigma \in L | \checkmark(p,\sigma) \wedge act(p,\sigma)]|}{|[\sigma \in L | act(p,\sigma)]|}$.

We use $L_2 = [\langle \triangleright, a, b, c, d, \square \rangle^{45}, \langle \triangleright, a, c, b, d, \square \rangle^{35}, \langle \triangleright, a, e, d, \square \rangle^{20}]$ to illustrate both place quality functions. Let us consider place $p_1 = (I_1, O_1)$ with $I_1 = \{a\}$ and $O_2 = \{b\}$. $score_{freq}(p_1, L_2) = score_{rel}(p_1, L_2) = {}^{80}/_{100} = 0.8$. For place $p_2 = (I_2, O_2)$ with $I_2 = \{a\}$ and $O_2 = \{b, e\}$: $score_{freq}(p_2, L_2) = score_{rel}(p_2, L_2) = 1$. Hence, the two scoring functions agree and show that the second place is a better candidate. Note that if the candidate place p does not inhibit any of the traces in the log, then both scores are 1 by definition.

In the remainder of this paper we will use $score_{rel}$ rather than $score_{freq}$. The choice can be motivated by considering event log $L_3 = [\langle c, d \rangle^{1000}, \langle a, b \rangle^{100}, \langle b, a \rangle^{10}, \langle a, a, a, a, \ldots, a \rangle]$ (with the last trace containing 1000 a's) and candidate place $p_1 = (I_1, O_1)$ with $I_1 = \{a\}$ and $O_2 = \{b\}$. $score_{freq}(p_1, L_3) = {}^{1100}/_{1111} = 0.99$, $score_{rel}(p_1, L_3) = {}^{100}/_{111} = 0.90$. Now the values are very different. Interpreting the scores reveals that $score_{freq}$ is too optimistic. Basically one can add any place connected to low frequent activities, without substantially lowering the

$score_{freq}$ score. Hence, $score_{rel}$ is preferable over $score_{freq}$. Based on the above discussion, we use the scoring function $score_{rel}$. in conjunction with a threshold t_{replay} to select the candidate places to be included in the Hybrid net (that is the candidate places with $score_{rel} > t_{replay}$).

Candidate places are explored in a particular order. This is needed because we may want to limit the number of places or time needed to construct them. Moreover, we will avoid adding conflicting or redundant places. In case of two conflicting places, one can only add the first one. Which one this is, depends on the order in which places are added.

Definition 26 (Ordering relation). *Let P_C be a set of candidate places. A function $ord : \{1, 2, \ldots, |P_C|\} \rightarrow P_C$ is an ordering relation over P_C iff $P_C = \{ord(i) \mid 1 \leq i \leq |P_C|\}$.*

Such an ordering could be based on $|I|+|O|$. This means that we first consider places having a few connecting arcs. It is always possible to start with a preorder (e.g., based on the number of arcs) and turn it into a total order. The ordering relation is used to consider candidate places in a particular order. Given the candidate places added already, we may consider discarding particular places. By doing this before evaluating the place quality, we can speed up discovery considerably.

Definition 27 (Exclusion relation). *Let P_C be a set of candidate places. $excl \subseteq P_C \times \mathcal{P}(P_C)$ is the exclusion relation. $(p, P) \in excl$ means that p is in conflict with a set of places P assuming exclusion relation $excl$.*

Different exclusion relations are possible. In the remainder, we consider an exclusion relation $excl$ based on the earlier place conflict definition (Definition 20) and place redundancy definition (Definition 21), i.e., $(p, P) \in excl$ if and only if $p \# P$ or $p \sim P$. This way we avoid adding redundant or conflicting places.

Algorithm 1 describes the details of the discovery technique. The algorithm iterates over the set of candidate places P_C in order to identify the places to be added to the set of places of the hybrid system net, until either the time threshold is over, the maximum number of internal places has been reached or all the candidate places have been inspected. Only the candidate places that reach the replay threshold (t_{replay}) and that do not meet the exclusion criteria are added (lines 4-8). Once the set of internal places P has been built, it is enriched with the start and end places (lines 9-11) and the set of normal, sure and unsure arcs built (lines 12-14).

Input:

- an event log $L \in \mathcal{U}_L$,
- a causal graph $G = (A, R_S, R_W) \in \mathcal{U}_G$,
- a candidate place set P_C,
- an ordering relation ord over P_C,
- an exclusion relation $excl$ over P_C,
- a threshold t_{replay} for evaluating candidate places,
- a threshold t_{max} for the maximal time allowed, and
- a threshold n_{max} for the maximal number of internal places.

begin
  $P \leftarrow \emptyset$
  $i \leftarrow 1$
  while $time < t_{max} \ \wedge \ i \leq |P_C| \ \wedge \ |P| < n_{max}$ **do**
    $p \leftarrow ord(i)$
    $i \leftarrow i + 1$
    if $score_{rel}(p, L{\restriction_A}) \geq t_{replay} \ \wedge \ (p, P) \notin excl$ **then**
      $P \leftarrow P \cup \{p\}$
  $p_{\triangleright} \leftarrow (\emptyset, \ \{\triangleright\})$
  $p_{\square} \leftarrow (\{\square\}, \ \emptyset)$
  $P \leftarrow P \cup \{p_{\triangleright}, \ p_{\square}\}$
  $F_I \leftarrow \{(a, p) \in A \times P \mid a \in \bullet p\} \cup \{(p, a) \in P \times A \mid a \in p\bullet\}$
  $F_{II} \leftarrow R_S \setminus \widehat{F_I}$
  $F_{III} \leftarrow R_W$

Output: a hybrid system net $SN = (PN, [p_{\triangleright}], [p_{\square}])$ with $PN = (P, A, F_I, F_{II}, F_{III})$.

Algorithm 1: Approach for Hybrid System Net Discovery.

It is easy to check that this concrete $disc_{hsn}$ function indeed ensures consistency. The construction of the discovered hybrid system net is guided by the causal graph. We can construct hybrid system net $disc_{hsn}(L, disc_{cg}(L))$ for any event

log L using parameters t_{freq}, c, w, t_{R_S}, t_{R_W}, and t_{replay}. Our discovery approach is highly configurable and also provides formal guarantees (e.g., $t_{replay} = 1$ ensures perfect fitness). When there is not enough structure or evidence in the data, the approach is not coerced to return a model that suggests a level of confidence that is not justified.

We applied our approach to the example event log shown in Figure 4 using the parameters $t_{freq} = 1000$, $c = 1$, $w = 0.2$, $t_{R_S} = 0.8$, $t_{R_W} = 0.75$ and $t_{replay} = 0.9$. In the first discovery step, the causal graph shown in Figure 10 was discovered, and, based on this causal graph, the hybrid Petri net shown in Figure 7 was discovered in the second discovery step. The causal graph contains two weak causal dependencies: (*place order* $\rightarrow$ *pay*) and (*send invoice* $\rightarrow$ *prepare delivery*). These weak dependencies are depicted as uncertain edges in the hybrid Petri net. The hybrid miner tries to discover formal places based on the strong causal dependencies of the causal graph. For example, the strong dependencies (*confirm payment* $\rightarrow$ *end*), (*make delivery* $\rightarrow$ *end*), and (*cancel order* $\rightarrow$ *end*) are transformed into the places $p7$ and $p8$ to formally model a choice between either canceling the order or confirming the payment and making the delivery. Other strong dependencies where the hybrid miner fails to discover formal places are depicted in the hybrid Petri net as sure edges, e.g., the sure edge (*send reminder* $\rightarrow$ *cancel order*).

6.3. Improving Performance

One of the reasons that commercial vendors resort to informal models or just the directly follow graph is performance. Process mining tools need to be able to deal with millions of events. We argue that adding formality and semantics do not need to create a major performance overhead and is feasible for larger data sets. This is consistent with the findings in Leemans et al. [28].

6.3.1. Limiting the number of candidate places.

The set of candidate places can be limited as described before (e.g., using P_C^k to ensure that places can have at most k connections). Users are seldom interested in places with many connections, as they are harder to understand and less precise. Pruning the set of candidate places has a dramatic effect on the search space with limited effects on the model's quality (see Section 8.1). The exclusion relation is another means to avoid considering candidate places.

6.3.2. Prematurely end place evaluation.

Valid places that are considered candidates and that are not excluded need to be evaluated. Computing $score_{rel}(p, L)$ is, in principle, linear in the size of the

log because all traces need to be replayed. However, often one does not need to traverse the whole log if it is already clear that the case do not pass the threshold. For example, when $t_{replay} = 0.9$ and we have already found problems for 10% of the cases, there is no need to check the rest. Therefore, we defined a *stopping criterion* that is safe in the sense that places of sufficient quality will be evaluated completely and places of insufficient quality are also classified correctly, but may be aborted earlier.

To explain, recall that $score_{rel}(p, L) = \frac{|\,[\sigma \in L | \checkmark(p,\sigma) \ \wedge \ act(p,\sigma)]\,|}{|\,[\sigma \in L | act(p,\sigma)]\,|}$. Now consider an event log L and candidate place p. Suppose that part of the event log, say L_1, has been processed and another part, say L_2, still needs to be processed. Note that $L = L_1 \uplus L_2$ (union of two multisets). We need to decide whether the statement $score_{rel}(p, L) \geq t_{replay}$ is true or not. This can be rewritten as

$$\frac{|\,[\sigma \in L_1 \mid \checkmark(p,\sigma) \ \wedge \ act(p,\sigma)]\,| + |\,[\sigma \in L_2 \mid \checkmark(p,\sigma) \ \wedge \ act(p,\sigma)]\,|}{|\,[\sigma \in L_1 \mid act(p,\sigma)]\,| + |\,[\sigma \in L_2 \mid act(p,\sigma)]\,|} \geq t_{replay}$$

Let us now assume the best possible scenario for the unseen part of the log (L_2), i.e., all remaining cases are fitting and activate the place.

$$|\,[\sigma \in L_2 \mid \checkmark(p,\sigma) \ \wedge \ act(p,\sigma)]\,| = |\,[\sigma \in L_2 \mid act(p,\sigma)]\,| = |\,L_2\,|$$

Therefore, the best possible situation is:

$$\frac{|\,[\sigma \in L_1 \mid \checkmark(p,\sigma) \ \wedge \ act(p,\sigma)]\,| + |\,L_2\,|}{|\,[\sigma \in L_1 \mid act(p,\sigma)]\,| + |\,L_2\,|} \geq t_{replay}$$

We know all these values already. Moreover, if the rewritten (optimistic) condition does not hold, it can never hold. Therefore, we can stop the evaluation of place p if

$$\frac{|\,[\sigma \in L_1 \mid \checkmark(p,\sigma) \ \wedge \ act(p,\sigma)]\,| + |\,L_2\,|}{|\,[\sigma \in L_1 \mid act(p,\sigma)]\,| + |\,L_2\,|} < t_{replay}$$

This means that for high t_{replay} values and poor candidates one only needs to inspect only few cases. In the extreme case where $t_{replay} = 1$ evaluation can stop on the first problem encountered.

6.3.3. Using a global place score.

In Definition 18, we defined two quality measures for the evaluation of candidate places: $score_{freq}(p, L)$ and $score_{rel}(p, L)$. We argued that $score_{rel}(p, L)$ is the better measure. In the worst-case scenario, one may need to traverse the whole event log to evaluate the quality of candidate place p. However, we can use a heuristic to prune the set of candidate places based on their expected score.

Definition 28 (Global Place Score). *Let $L \in \mathcal{U}_L$ be an event log. For any place $p = (I, O)$, we define the following score: $score_{glob}(p, L) = 1 - \frac{|\#(I,L) - \#(O,L)|}{max(\#(I,L), \#(O,L))}$ is a global score only looking at the aggregate frequencies of activities.*

The intuition is that $score_{glob}(p, L)$ correlates positively with $score_{rel}(p, L)$. If all traces fit perfectly, then $score_{glob}(p, L) = 1$. To illustrate the idea, consider $L_2 = [\langle \triangleright, a, b, c, d, \square \rangle^{45}, \langle \triangleright, a, c, b, d, \square \rangle^{35}, \langle \triangleright, a, e, d, \square \rangle^{20}]$ and place $p_1 = (I_1, O_1)$ with $I_1 = \{a\}$ and $O_2 = \{b\}$. $score_{rel}(p_1, L_2) = {}^{80}/_{100} = 0.8$ and $score_{glob}(p_1, L_2) = 1 - {}^{|100-80|}/_{max(100,80)} = 0.8$. For place $p_2 = (I_2, O_2)$ with $I_2 = \{a\}$ and $O_2 = \{b, e\}$: $score_{rel}(p_2, L_2) = score_{glob}(p_2, L_2) = 1$. Although the scoring functions agree in this example, it is clear that the global place score is only an approximation. This can be illustrated using the event log $L_3 = [\langle c, d \rangle^{1000}, \langle a, b \rangle^{100}, \langle b, a \rangle^{10}, \langle a, a, a, a, \ldots, a \rangle]$ (with the last trace containing 1000 a's) and candidate place $p_1 = (I_1, O_1)$ with $I_1 = \{a\}$ and $O_2 = \{b\}$. $score_{rel}(p_1, L_3) = {}^{100}/_{111} = 0.90$, $score_{glob}(p_1, L_3) = 1 - {}^{|1110-110|}/_{max(1110,110)} = 0.099$. Now the values are very different. This is caused by the extreme trace having 1 000 a's. Clearly, the approximation can be further improved by bounding the activity count per case.

The value of $score_{glob}(p, L)$ can be computed very efficiently because traces do not need to be replayed and can therefore be used to quickly prune the set of candidate places. However, as the example shows, one needs to be careful when traces are unbalanced (i.e., I or O activities occur many times in a few traces). A global score threshold t_{glob} can be set and used, when needed, to decide the candidate places to be quickly discarded based on their $score_{glob}(p, L)$ value.

6.3.4. Trace variant representation.

Real-life event logs often follow a Pareto distribution of traces [31]. In most event logs a large fraction of behavior is covered by a small fraction of unique activity sequences (*trace variants*) that occur frequently. It would hence be sufficient to replay each trace variant once when evaluating a candidate place, so as to improve time performance for logs that follow a Pareto distribution of traces with no effect on the quality of the discovered models. For example, consider the example event log shown in Figure 4. This event log consists of 12 666 cases that can be grouped into nine unique activity sequences. When evaluating a candidate place, we can achieve a large improvement in time by replaying the nine trace variants instead of replaying all 12 666 traces.

We create for the input event log a *trace variant representation*; i.e., we transform the log into a list of unique activity sequences with their frequencies. This

actually corresponds to our definition of event logs as a multiset of activity sequences (Definition 1), and it also conforms with the trace variant visualization we used in Figure 4 to represent our example event log. This compact representation is used instead of the original log where each case is represented and checked separately for helping to speed up the following processes:

(i) In the step of causal graph discovery: we compute all metrics needed to construct the causal graph based on the trace variant representation instead of using the original traces. We then multiply the obtained metrics by the corresponding trace variant frequency.

(ii) In the process of evaluating candidate places: we compute the fitness score for each sequence of activities only once. We then multiply the obtained score by the corresponding trace variant frequency.

This idea may lead to large time improvements for huge logs containing trace variants with high frequencies. Assume we have an event log of 10 trace variants each with a frequency of 1 000 and a set of valid candidate places of size n, we are able to reduce the number of performed place evaluations from $10\,000 \times n$ to $10 \times n$.

6.3.5. Evaluating maximal places before smaller candidate places.

There are many possible ordering relation strategies (see Definition 26) that can be used for building and evaluating candidate places. Experience shows that it may be beneficial to check so-called *maximal places* first. Maximal places are generated by combining sure edges in an iterative way, so that every two sure edges sharing the same input node or the same output node are covered by the same maximal place.

Definition 29 (Maximal Place). *Let $G = (A, R_S, R_W) \in \mathcal{U}_G$ be a causal graph. $R \subseteq R_S$ is a* cluster *of sure arcs if (1) $R \neq \emptyset$, (2) $\forall_{(a_1,a_2)\in R}\ \forall_{a\in A}\ \{(a_1, a), (a, a_2)\} \cap R_S \setminus R = \emptyset$, and (3) there is no smaller $R' \subset R$ satisfying the same properties. $p_R \in \mathcal{P}_A$ is a* maximal place *if there is a cluster R such that $p_R = (\{a_1 \mid (a_1, a_2) \in R\}, \{a_2 \mid (a_1, a_2) \in R\})$*

Given a set of sure arcs R_S, these arcs can be partitioned into a set of clusters. Two different clusters cannot have two arcs with the same input or output activity (they would be merged). Clusters need to be minimal, i.e., the partitioning is maximal. For each cluster R, there is one maximal place p_R.

For example, let us assume we have a causal graph with a set of sure edges $R_S = \{(a, c), (b, c), (b, d), (e, f)\}$. The cluster that contains the first sure edges (a, c) must also contain (b, c) since both edges share the same output node c. The cluster must also contain (b, d) because (b, c) and (b, d) share the same input node b. Therefore, R_S will be partitioned into the two clusters: $R_1 = \{(a, c), (b, c), (b, d)\}$ and $R_2 = \{(e, f)\}$. Based on these two clusters, the following two maximal places will be generated: $p_{R_1} = (\{a, b\}, \{c, d\})$ and $p_{R_2} = (\{e\}, \{f\})$. Note that a maximal place does not need to be valid. For example, p_{R_1} indicates a dependency from a to d although this dependency does not correspond to a sure edge (i.e., $(a, d) \notin R_S$).

Before building the candidate places according to one of the ordering relationship strategies, we check the maximal places. If any of these places is valid and achieves the minimally required place evaluation score, then the place is added to the hybrid Petri net, thus allowing the algorithm to stop evaluating further places for that set of edges and hence sparing time.

6.3.6. Distribution

Although this is not the topic of this paper, the time-consuming steps of the approach can also be distributed. The construction of the causal graph requires counting how many times some activity a was followed by activity b. By partitioning the event log L in n smaller sublogs L_i such that $L = L_1 \uplus L_2 \uplus \ldots \uplus L_n$, these numbers can be counted per sublog and easily aggregated. The same holds for the evaluation of the place quality. Big data platforms like Apache Hadoop, Spark, and Flink can be used for this [32].

7. Implementation

Two ProM plug-ins have been created to support the approach described in this paper:[3] the *Causal Graph Miner* and the *Hybrid Petri Net Miner*.

The *Causal Graph Miner* plug-in is used to create a causal graph using the approach described in Definition 9. The user can initially set the values of the parameters t_{freq}, t_{R_S}, t_{R_W} and w (see Figure 12 on the left)[4]. The initial causal graph will be created based on these initial values of the parameters. The user can then control the values of the parameters interactively through sliders and directly see the effects in the resulting graph (see Figure 13 on the left). The sure and unsure

[3] Install ProM and the package *HybridMiner* from `http://www.promtools.org`.

[4] Some default values are provided based on the parameter evaluation carried out in Section 8.6.

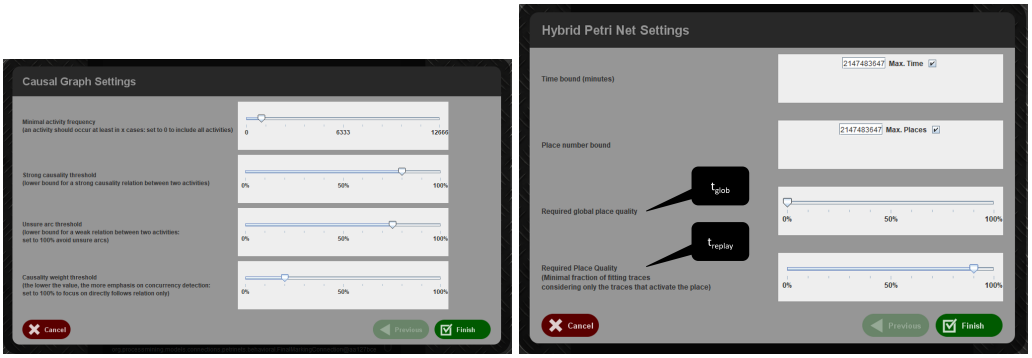


Figure 12: Screenshots of the *Causal Graph Miner* (left) and the *Hybrid Petri Net Miner* (right) setting parameters.

edges of the hybrid causal graph are depicted with different colors (blue and red respectively in Figure 13) and the causal graph is annotated with the causal metrics $Caus_{c,w}$. The user can select the metrics to visualize: besides $Caus_{c,w}$, the number of direct successions ($\#(a, b, L)$), the percentage of incoming and outgoing direct successions ($\frac{\#(a,b,L)}{\#(*,b,L)}$ and $\frac{\#(a,b,L)}{\#(a,*,L)}$, respectively), the strength of the relation taking into account the split and join behavior ($Rel1$) as well as the strength of the relation considering concurrency and loops ($Rel2_c$). In order to make the visualization more user-friendly, besides the slider and the panel enabling the choice of the metrics to be visualized, the plug-in has been equipped with the possibility of changing the colors of the different types of edges, with the capability of exporting the visualization, as well as with the possibility of zooming in/out.

The *Hybrid Petri Net Miner* plug-in implements Algorithm 1. It takes as input the hybrid causal graph and returns a discovered hybrid system net. Only places that meet the t_{glob} and the t_{replay} thresholds are added. The replay approach has been optimized to stop replaying a trace when it does not fit. The user sets the values of the parameter t_{glob} and t_{replay} and a few other configuration parameters, thereby limiting the search for the best set of places for the discovered hybrid Petri net (i.e., time and place number bound), as shown in Figure 12 (right). The plugin also provides the possibility of changing the colors of the different types of arcs and places, exporting the visualization, as well as zooming in/out.

Figure 13 shows the two plug-ins in action for the event log containing $12\,666$ cases and $80\,609$ events. The results returned correspond to the causal graph depicted in Figure 10 and the hybrid system net depicted in Figure 7. Both were computed in less than a second on a standard laptop. Activity $send\,reminder$ may occur repeatedly (or not) after sending the invoice but before payment or cancel-

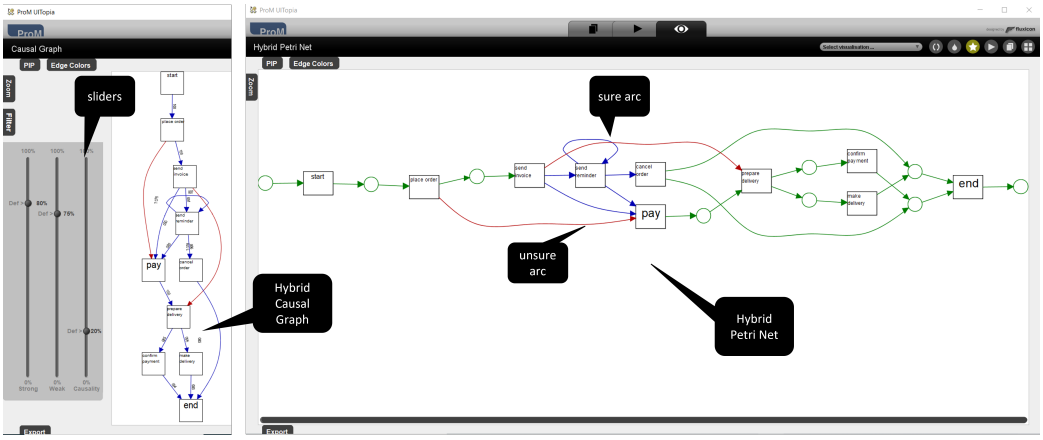


Figure 13: Screenshots of the *Causal Graph Miner* (left) and the *Hybrid Petri Net Miner* (right) analyzing the log containing 12 666 cases and 80 609 events of the example described in Section 2.4 with parameter settings $t_{freq} = 1000$, $c = 1$, $w = 0.2$, $t_{R_S} = 0.8$, $t_{R_W} = 0.75$, $t_{glob} = 0$ and $t_{replay} = 0.9$.

lation. However, payments may also occur before sending the invoice. The hybrid system net in Figure 13 (also see Figure 7 which is better readable) clearly differentiates between (1) the behavior which is dominant and clear and (2) the more vague behavior that cannot be captured formally or is not supported by enough "evidence".

8. Evaluation

In this section, we evaluate the *Causal Graph Miner* and the *Hybrid Petri Net Miner* plug-ins that combined implement the Hybrid Miner approach. The plug-ins are evaluated both in terms of performance, as well as the quality of the discovered models. We want to compare the results of the version of the Hybrid Miner described in this paper (HM_NV) with the ones obtained with the previous version of the Hybrid Miner described in [29] (HM_OV), as well as with existing state-of-the-art process discovery approaches.

As mentioned, process discovery techniques are typically evaluated using indicators referring to traditional Petri net-based *recall*, *precision*, *generalization*, and *simplicity* [1]. Although these metrics can give us an idea of the differences between hybrid Petri nets, given our goal to discover hybrid models explicitly capturing vagueness while providing solid guarantees, these classical measures are less suitable - especially when the goal is to compare with classical process

discovery approaches, such as the *Inductive Miner* [26]. To this aim, we evaluate the quality of the models discovered not only on classification tasks (that is, their capability to accept traces that are assumed to be compliant and to discard traces that are assumed to be non-compliant) but also by looking at strong and weak causal-graph-based precision and recall metrics $SCR(L, M)$, $SCP(L, M)$, $WCR(L, M)$, and $WCP(L, M)$ described in Section 4.3. Finally, we are interested to investigate the impact of different parameter values on the discovered models.

Specifically, we would like to address the following research questions:

RQ1. What differences do exist in terms of time performance and quality (measured through classical metrics and qualitative analysis) of the discovered models between HM_OV and HM_NV?

RQ2. How does the Hybrid Miner algorithm behave with respect to state-of-the-art discovery algorithms (using the Inductive Miner as a reference benchmark)?

RQ3. How do different parameter values affect the time performance of HM_NV and the quality of the discovered models?

In order to answer **RQ1**, we compare the performance (in terms of execution time) and the quality (in terms of classical process mining metrics and supported by a qualitative analysis) of the discovered models on three different sets of logs. The first set of logs is composed of real-life logs. The second set of logs aims at evaluating the performance of HM_OV and HM_NV on event logs while controlling log features such as size and other characteristics. The third set of logs focuses on noisy and infrequent behavior.

Section 8.1 summarizes our findings based on an analysis involving six data sets taken from the well-known *BPI Challenges* [33] by focusing on classical Petri net recall and precision measures. Section 8.2 reports in particular on the performance characteristics of the Hybrid Miner. We applied our approach to a large number of synthetic event logs with different sizes and generated by models with different characteristics. We use synthetic event logs because for these logs we have a known ground truth and can easily vary characteristics in a controlled manner. In Section 8.3 we show the effects of the Hybrid Miner on concurrency, duplicates, skips, and non-local dependencies. These experiments show that hybrid models can still provide guarantees and additional insights, even when the underlying process behavior is at odds with the representational bias of the discovery approach.

In order to answer **RQ2**, we carry out three different types of analysis. We first compare the Hybrid Miner and existing approaches (in particular the Inductive Miner) on the *BPI Challenges* [33] in Section 8.4. By ignoring the sure (*Type II*) and unsure (*Type III*) edges, we obtain regular Petri nets and we can use classical Petri net-based recall and precision measures. In the same section, strong and weak causal-graph-based precision and recall metrics (see Section 4.3) are also computed on the same logs so as to take into account also sure and unsure edges. Finally, in Section 8.5, we also use the *Process Discovery Contest* logs that turn process discovery into a classification problem. This allows us to compare the two versions of our approach with the Inductive Miner producing formal models without penalizing the Hybrid Miner because of informal dependencies.

Finally, Section 8.6 aims at answering **RQ3**. It shows the impact of different values of the Hybrid Miner parameters (t_{freq}, t_{R_S}, w, t_{replay}) on the results.

All the experiments reported in this evaluation have been carried out on a workstation Dell Precision 7820 with the following configuration: 314GB of RAM, double Intel Xeon Gold 6136 3.0GHz CPUs, and a 256GB SATA Solid State Drive.

8.1. BPI Challenge Logs

In this section, we evaluate the Hybrid Miner on a large number of real-life data. For this part of the evaluation, we used the same six real-life BPI Challenge data sets that we used to evaluate the initial version of the Hybrid Miner in [29] (HM_OV): BPI Challenge 2011 [34], BPI Challenge 2012 [35], BPI Challenge 2017 [36], activity log for incidents of the BPI Challenge 2014 [37], municipality 1 log of the BPI Challenge 2015 [38], and a subset of the BPI Challenge 2016 log for logged-in clicks [39]. Table 1 shows an overview of the six event logs: *BPI20XX* refers to the year of the corresponding BPI Challenge and the number of cases, events, and unique activities (event classes) are shown. We tested different values for the parameters, and we selected the values that led to the best trade-off between fitness and precision. The values selected for the parameters are reported in Table 2[5].

We compared the differences between HM_OV and HM_NV in terms of time performance and quality of the discovered models. The results of the evaluation are shown in Table 3. For each log, we discovered two models (a model using

[5]In all the evaluations carried out in this paper, we always use $t_{glob} = 0$, no bound on the number of places, a candidate place ordering relation strategy going from small to large places, but evaluating maximal places first.

Log	Number of cases	Number of variants	Number of events	Number of activities
BPI2011	1 143	981	150 291	624
BPI2012	13 087	4 336	164 506	23
BPI2014	46 616	22 632	466 737	39
BPI2015	1 199	1 170	52 217	398
BPI2016	557	557	286 075	312
BPI2017	31 509	5 623	475 306	24

Table 1: BPI Challenge event logs used for the evaluation.

Log	t_{freq}	t_{R_S}	t_{R_W}	w	t_{replay}
BPI2011	650	0.5	0.5	0.5	0.5
BPI2012	5 000	0.3	0.3	0.5	0.7
BPI2014	15 000	0.5	0.5	0.5	0.7
BPI2015	720	0.5	0.5	0.5	0.6
BPI2016	445	0.5	0.5	0.1	0.8
BPI2017	5 000	0.1	0.1	0.5	0.7

Table 2: Parameters used for evaluating the BPI Challenge logs.

Log	$\|P\|$		Fitness		Precision		Time (milliseconds)	
	HM_OV	HM_NV	HM_OV	HM_NV	HM_OV	HM_NV	HM_OV	HM_NV
BPI2011	10	10	0.552	0.552	0.111	0.111	11 916 165	**4 286**
BPI2012	10	10	0.8147	**0.815**	0.317	0.317	23 453	**4 165**
BPI2014	8	8	0.85	0.85	0.667	0.667	38 677	**19 725**
BPI2015	42	40	0.499	0.499	**0.892**	0.839	636 937	**3 867**
BPI2016	2	2	0.757	0.757	0.097	0.097	49 481	**3 553**
BPI2017	34	29	0.908	**0.912**	**0.527**	0.486	2 824 595	**14 143**

Table 3: Results of the BPI Challenge evaluation.

HM_OV and another model using HM_NV) and reported the number of discovered places, the fitness and precision values, as well as the time required to discover each model, by highlighting in bold the highest values for fitness and precision and the lowest ones for the required time.

The results show that HM_NV is able to achieve major improvements in terms of time performance. The quality of the discovered models, in terms of classical fitness and precision metrics, is, however, not affected. For four out of the six logs, both versions of the Hybrid Miner discovered the exact same models, but the new version (HM_NV) is significantly faster. For example, HM_NV discovered a model for BPI2011 in approximately four seconds compared to more

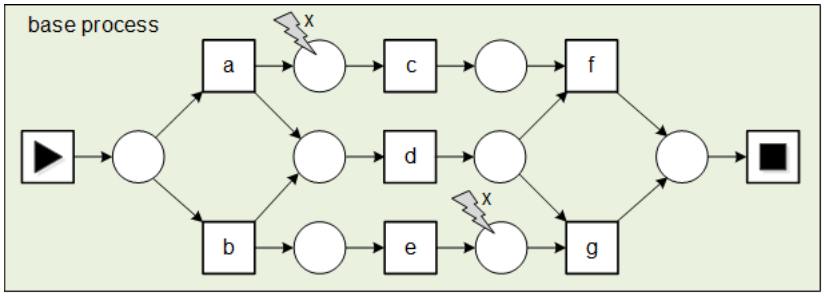


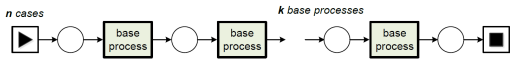
Figure 14: Base model for the generation of the event logs used for testing the scalability results [40].

than three hours required by the old version (HM_OV) to discover the same models. There are two cases where the quality metrics are slightly worse (BPI2015 and BPI2017), because the new version discovered fewer places than the old one leading to less precise models. For example, HM_NV discovered 29 places for BPI2017 compared to 34 places discovered by HM_OV. This decreased the precision from 0.527 to 0.486 while increasing fitness. This is due to post-processing where we remove places connecting overlapping sets of transitions. We consider the decrease in precision for BPI2015 and BPI2017 to be acceptable considering the huge improvements in time.

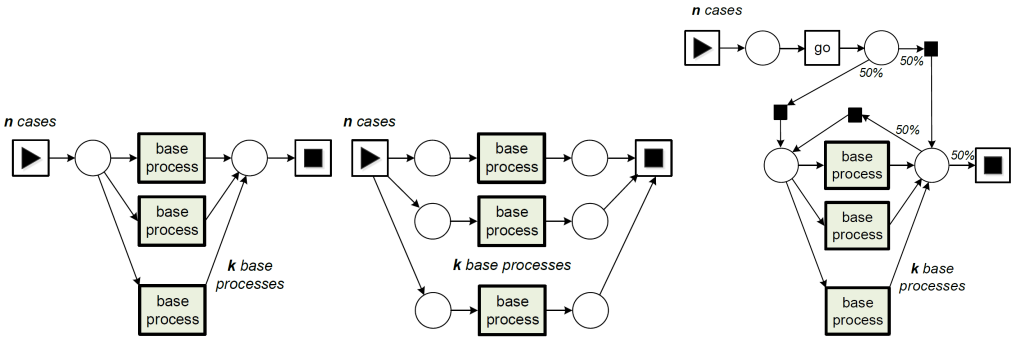
8.2. Scalability Analysis Using Parameterized Event Logs

In this section, we focus on comparing HM_OV and HM_NV especially in terms of the capability to efficiently deal with a collection of synthetic logs [40] with increasing size and complexity. The collection of logs is generated by combining different replicas of the same model (reported in Figure 14) through different constructs (SEQUENCE, CHOICE, PARALLEL and LOOP) and with different event log sizes. The constructs used to combine replicas of the base model are shown in Figure 15. We are indeed interested in evaluating the performance of the approach with different constructs, different trace lengths (obtained with different replicas of the model) and different sizes of the event log. As shown in Figure 14, each base component covers seven main activities and an additional activity that can happen at any point in between them. For each type of model, different numbers of replicas (1, 2, 5, 10 and 100) and hence different sizes of the model alphabet (8, 16, 40, 80 and 800) have been considered. Replicated transitions are denoted as a_1, b_1, a_2, b_2, etc.

Moreover, for a fixed number of replicas (2 and 10), data sets of different sizes have been generated (1 000, 10 000 and 100 000 cases). For the models



(a) A sequence of k base models.



(b) A choice of k base models. (c) A concurrency of k base models. (d) A loop of k base models.

Figure 15: Constructs used for the generation of the event logs used for testing the scalability results [40].

Replica #	Alphabet size	Number of cases	PARALLEL		SEQUENCE		CHOICE		LOOP	
			t_{R_S}	t_{R_W}	t_{R_S}	t_{R_W}	t_{R_S}	t_{R_W}	t_{R_S}	t_{R_W}
1	8	1 000	0.5	0.5	0.5	0.3	0.5	0.3	0.2	0.2
2	16	100, 1 000, 10 000	0.5	0.5	0.5	0.3	0.5	0.3	0.2	0.2
5	40	1 000	0.5	0.5	0.5	0.3	0.5	0.3	0.2	0.2
10	80	1 000, 10 000, 100 000	0.4	0.4	0.5	0.3	0.5	0.3	0.2	0.2
100	800	10 000, (100 000)	0.4	0.4	0.5	0.3	0.1	0.1	0.2	0.2

Table 4: Alphabet size and the number of cases, as well as values used for the parameters t_{R_S} and t_{R_W} in the scalability evaluation experiments for different numbers of replicas of the base components.

composed of 1 and 5 replicas, only the data set with 1 000 cases has been created, while for the models composed of 100 replicas, data sets with 10 000 and 100 000 cases have been generated for CHOICE and LOOP constructs, and only data sets with 10 000 cases have been considered for SEQUENCE and PARALLEL constructs. The first three columns of Table 4 summarize the data set composition, that is, for each number of replicas, the corresponding alphabet size and the size in terms of the number of cases of the generated data sets are reported. The logs are labeled by combining the type of composition, the number of replicas, and the number of traces. For example, the log CHOICE_5_10000 records the execution of 10 000 cases of the process generated by combining five replicas of the base model through a CHOICE construct.

For comparing HM_OV and HM_NV on the event logs, we fixed the values of

some of the parameters ($t_{freq} = 0$, $w = 0.5$, $t_{replay} = 0.9$), while for t_{R_S} and t_{R_W}, variable values are used - depending on the number of base components. For the PARALLEL and CHOICE sets, we used lower thresholds for logs with more base components in order to discover optimal models. This is necessary because the dependency scores for the causal relations between activities decrease after adding more base components. The values used for the thresholds for sure and uncertain causal edges (t_{R_S} and t_{R_W}, respectively) are reported in the last columns of Table 4.

The complete results of this experiment are reported in Table 5. Two models were discovered for each log (HM_OV and HM_NV); for each model, we report the number of places, fitness, precision, and the total time required to discover the model, by reporting in bold the highest fitness/precision and the lowest time. We set a timeout threshold to 3 hours and we report a timeout (T) when the computation exceeds the timeout. In the following, we first focus on the time performance, by investigating the effect of increasing the log size (i.e., the number of cases) and the number of replicas on the time performance. Then, we discuss the quality of the discovered models.

Figure 16 shows, for each of the considered model structures, the different time performance obtained by running the Hybrid Miner on the data set built with 10 different replicas for 1 000, 10 000 and 100 000 traces.

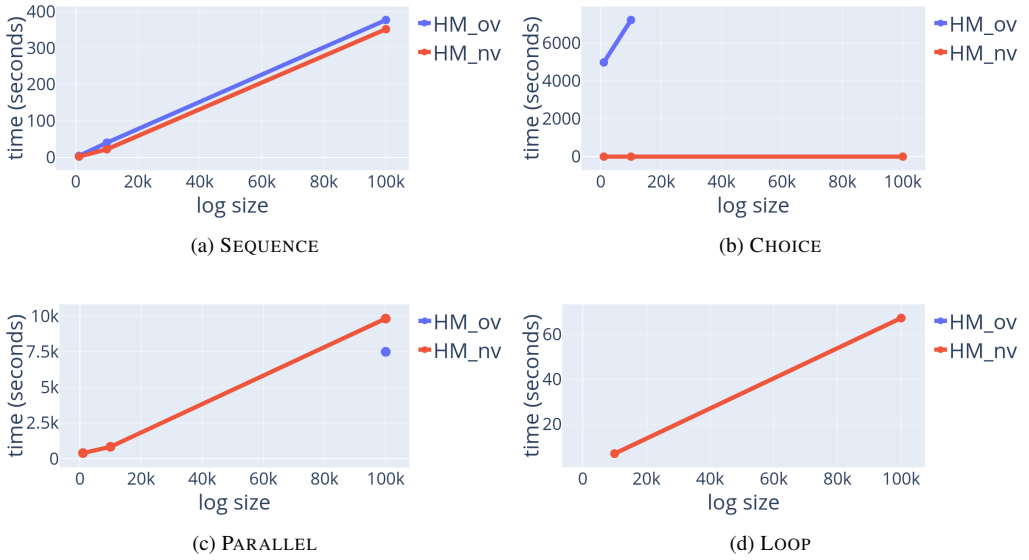
The points related to the executions exceeding the timeout are not reported in the plots. The plots clearly show that HM_NV performs overall better than HM_OV, which often ends up in a timeout. Moreover, as expected, the required time increases together with the log size, and the required time is higher for models in which the parallelization is high, while it is lower for event logs generated from models built by using the CHOICE construct. Nevertheless, the figure also reveals some unexpected results. The first one is related to the LOOP logs: HM_NV, indeed, is able to discover models very fast for logs with 10 000 and 100 000 cases, while it reaches the timeout for the data set of size 1 000. This could be due to the fact that the noise in the log of size 1 000 does not allow the fast discovery of proper maximal places that, instead, are identified more easily for the logs having a data set size of 10 000 and 100 000. A second unexpected behavior is related to the ability of HM_OV to discover models with the PARALLEL event log with 100 000 cases and not with smaller event logs having 1 000 and 10 000 cases. This could be possibly due to the fact that the model that HM_OV is able to discover with the log of size 100 000 actually has several missing places (around 20 less of the places of the model discovered by HM_NV).

Figure 17 shows instead, for each of the considered model structures, the dif-

Structure	Replica #	Number of cases	Place #		Fitness		Precision		Time (s)	
			HM_OV	HM_NV	HM_OV	HM_NV	HM_OV	HM_NV	HM_OV	HM_NV
SEQUENCE	1	1 000	10	10	0.995	**1.000**	0.533	**0.633**	0.620	**0.556**
	2	100	17	17	1.000	1.000	0.432	0.432	0.647	**0.630**
	2	1 000	17	17	1.000	1.000	0.468	0.468	0.841	**0.690**
	2	10 000	17	17	0.993	0.993	1.000	1.000	3.001	**0.759**
	5	1 000	38	38	1.000	1.000	0.272	0.272	1.588	**1.067**
	10	1 000	73	73	0.999	0.999	0.169	0.169	3.939	**1.897**
	10	10 000	73	73	0.990	0.990	0.897	0.897	40.170	**22.132**
	10	100 000	73	73	0.990	0.990	0.949	0.949	376.968	**351.702**
	100	10 000	703	703	0.990	0.990	-	-	2558.627	**1 123.704**
CHOICE	1	1 000	10	10	1.000	1.000	0.635	0.635	0.725	**0.687**
	2	100	16	16	1.000	1.000	0.434	0.434	0.769	**0.678**
	2	1 000	16	16	1.000	1.000	0.478	0.478	1.691	**1.570**
	2	10 000	16	16	0.994	0.994	1.000	1.000	2.510	**0.798**
	5	1 000	34	34	1.000	1.000	0.340	0.340	5.407	**1.334**
	10	1 000	64	64	1.000	1.000	0.280	0.280	4979.517	**2.050**
	10	10 000	85	64	0.320	**1.000**	0.172	**0.307**	7 220.501	**1.582**
	10	100 000	-	64	-	**0.994**	-	**1.000**	T	**2.145**
	100	10 000	-	604	-	**0.999**	-	**0.264**	T	**19.941**
	100	100 000	-	604	-	**0.994**	-	**0.855**	T	**20.849**
PARALLEL	1	1 000	10	10	0.994	**1.000**	0.533	**0.634**	0.701	**0.527**
	2	100	18	18	1.000	1.000	0.431	0.431	0.693	**0.631**
	2	1 000	18	18	0.997	**1.000**	0.513	**0.525**	1.704	**1.342**
	2	10 000	18	18	0.993	0.993	1.000	1.000	3.911	**1.330**
	5	1 000	42	42	1.000	1.000	0.195	0.195	11.647	**1.736**
	10	1 000	-	82	-	**1.000**	-	**0.089**	T	**390.108**
	10	10 000	-	82	-	**0.990**	-	**0.199**	T	**831.414**
	10	100 000	62	82	0.981	**0.991**	0.090	**0.228**	**7 489.813**	9820.529
	100	10 000	-	-	-	-	-	-	T	T
LOOP	1	1 000	10	10	0.992	0.992	0.954	0.954	0.890	**0.638**
	2	100	16	16	0.996	**1.000**	**0.519**	0.409	2.252	**1.607**
	2	1 000	16	16	0.993	0.993	0.881	0.881	2.859	**1.080**
	2	10 000	16	16	0.993	0.993	0.934	0.934	22.648	**1.512**
	5	1 000	-	34	-	**0.995**	-	**0.483**	T	**4.356**
	10	1 000	-	-	-	-	-	-	T	T
	10	10 000	-	64	-	**0.992**	-	**0.752**	T	**6.822**
	10	100 000	-	64	-	**0.992**	-	**0.805**	T	**67.255**
	100	10 000	-	-	-	-	-	-	T	T
	100	100 000	-	-	-	-	-	-	T	T

Table 5: Scalability evaluation results.

ferent time performance obtained by running HM_OV and HM_NV on the data set built with 2, 10 and 100 replicas of the model and with size 10 000. Also in this setting, we aborted computations exceeding the 3 hours. Also these plots clearly show that HM_NV performs overall better than HM_OV, which often ends up in a timeout. However, also HM_NV exceeds the timeout with event logs generated by models with 100 replicas for the PARALLEL and for the LOOP structures. Moreover, as expected, the required time increases together with the number of replicas, and the required time is much higher for models in which there is a lot of concurrency, while it is lower for event logs generated from models built by using



(a) SEQUENCE (b) CHOICE

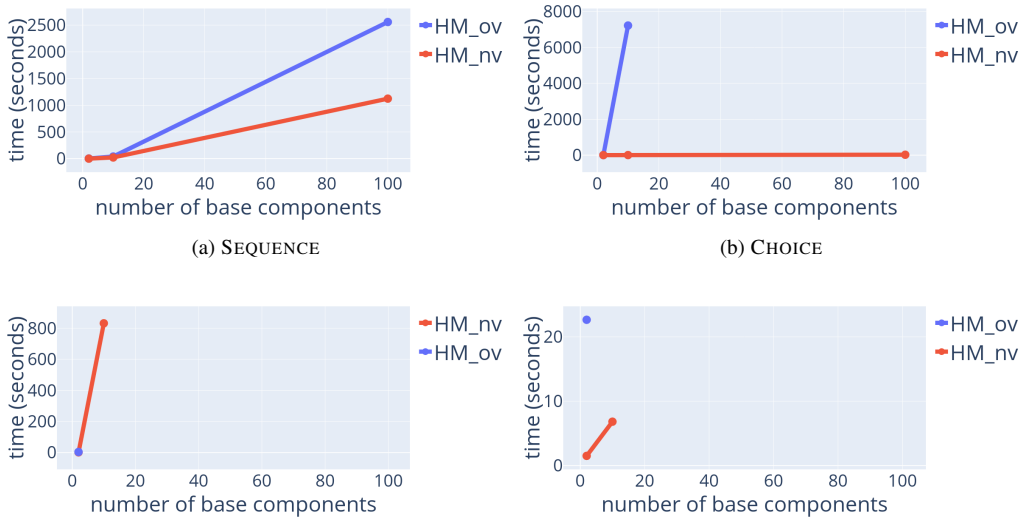
(c) PARALLEL (d) LOOP

Figure 16: Time performance for different event log sizes (i.e., the number of cases).

the CHOICE construct. Also in this setting, for the LOOP, when the Hybrid Miner is able to return results, they are returned in a short time, which is mainly due to the identification of correct maximal places.

Moving to the quality of the discovered models, we can observe that for most of the logs for which we did not obtain a timeout exception, both versions of the Hybrid Miner delivered identical models. However, we observe some exceptions. Let us consider the log CHOICE_10_10000, that is the event log of size $10\,000$ built starting from 10 replicas of the base model composed via the CHOICE construct. For this log, HM_NV shows a decrease in the number of places from 85 to 64 with respect to HM_OV. This log contains an XOR-choice between ten base components. The improvement of evaluating maximal places before building smaller places led to the representation of this choice through two places (a place for the XOR-join and a place for the XOR-split). On the other hand, HM_OV was not able to discover these maximal places and replaced them with 23 places; each of these 23 places meets the minimal required local fitness score ($t_{replay} = 0.9$), but together, they lead to a global fitness score of 0.32. Figure 18 shows the XOR-split discovered by HM_OV and HM_NV.

Another interesting example is the log PARALLEL_10_100000. For this log, we can observe an increase in the number of places of HM_NV with respect to



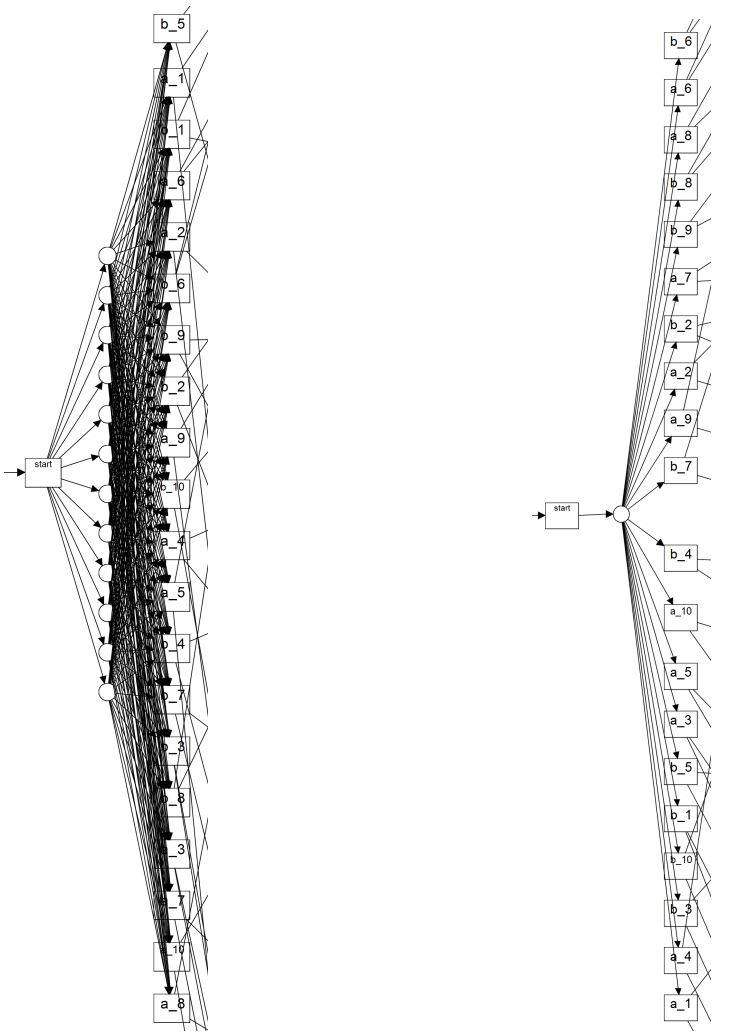
(a) SEQUENCE (b) CHOICE

(c) PARALLEL (d) LOOP

Figure 17: Time performance for replicated process constructs (1, 2, 10 and 100 times) resulting in event logs with vastly different alphabet sizes (ranging from 8 activities to 800 unique activities). Missing values correspond to computations aborted after three hours.

HM_OV from 62 to 82, and an increase in both fitness and precision. This log, indeed, exhibits parallelism between the ten base components where each base component contains, in turn, an XOR-choice. Evaluating maximal places before building smaller places led to the optimal representation of this complex structure through 20 places (10 places for the AND-join and 10 places for the AND-split). On the other hand, HM_OV was not able to discover any of these 20 maximal places and represented the parallelism informally instead. Figure 19 shows the representation of the AND-split by HM_OV and HM_NV.

Finally, the last difference between the models discovered by HM_OV and HM_NV is the representation of noise. For instance, Figure 20 shows the models discovered for the log SEQUENCE_1_1000. In this log, activity x_1 represents noise. We observe that HM_NV represented noise informally, while x_1 is formally connected to c_1 through a place in the model discovered by HM_OV, i.e., $p = (\{a\_1, x\_1\}, \{c\_1\})$. The informal representation of noise allows HM_NV getting higher fitness and precision values with respect to HM_OV, while executing activity x_1 in the model discovered by HM_OV always generates remaining tokens that can never be consumed, so that all traces containing noise do not fit the model.



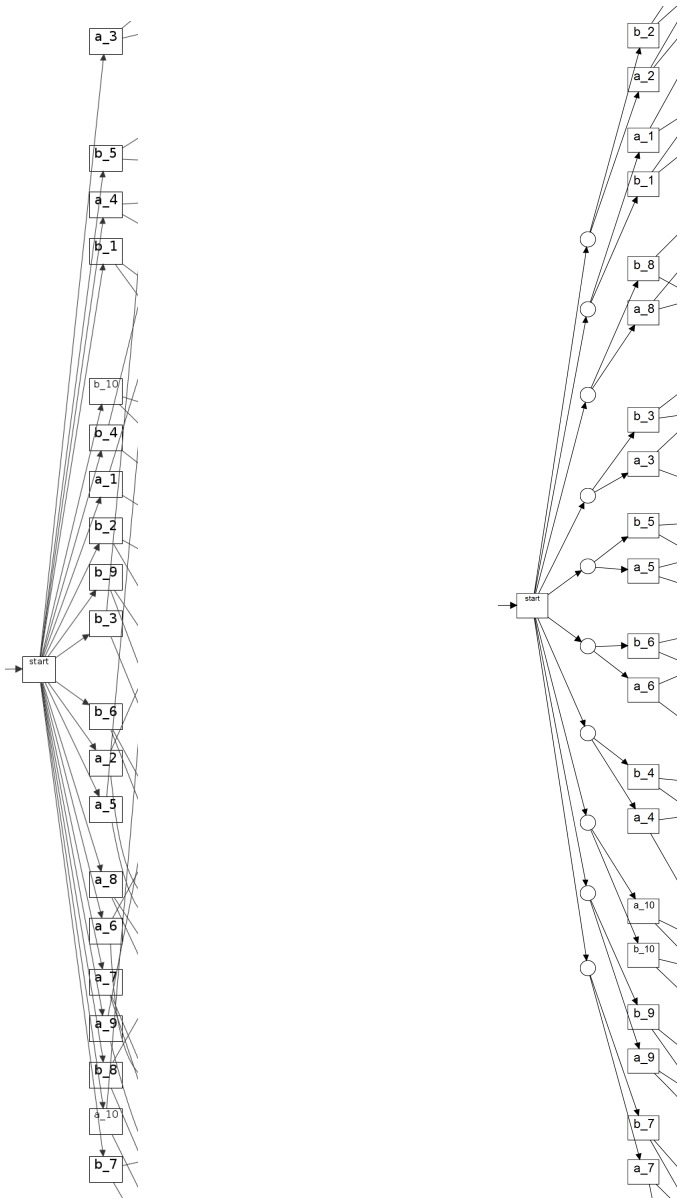
(a) Representation by HM_OV.

(b) Representation by HM_NV.

Figure 18: The representation of an XOR-split between 20 activities after the activity "start". Each of the places discovered by HM_OV meets to required local fitness score (0.9). However, the combination of places leads to an extremely low overall fitness score (0.32). The HM_NV performs much better and returns only the maximal place (instead of discovering many places with reasonable, but not optimal, fitness scores).

8.3. Handling Concurrency, Duplicates, Skips, and Non-Local Dependencies

In this subsection, we evaluate HM_OV and HM_NV when we vary representational biases together with noisy and infrequent behavior. To do this, we leverage a collection of $120 = 4 \times 6 \times 5$ synthetic logs [41] generated by models charac-



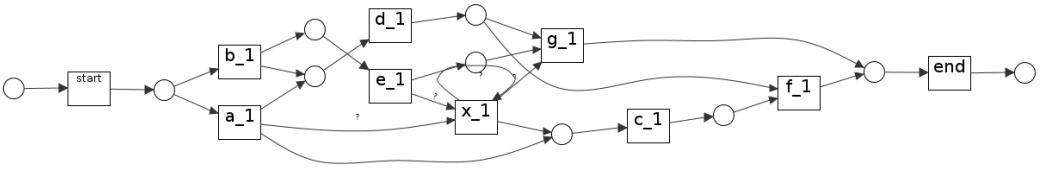
(a) Representation by HM_OV.

(b) Representation by HM_NV.

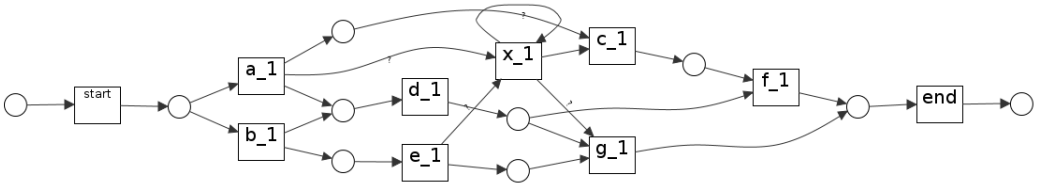
Figure 19: The representation of an AND-split of 10 base components following activity "start". Each base component starts with an XOR-split making a choice between two activities.

terized by four constructs[6]:

[6] Figure 21 allows for getting a grasp of the four constructs - although they are not the original models but the discovered ones.



(a) Model discovered by HM_OV.



(b) Model discovered by HM_NV.

Figure 20: The models discovered for the log SEQUENCE_1_1000.

- `parallel`: a simple parallel structure among five activities a, b, c, d and e; all five activities must be executed, but they can be executed in any order.
- `skip`: a simple skip structure, i.e., a sequence of two activities (a and b) with an additional optional activity c that can be executed in between. This model allows for only two trace variants: $\langle a, b, c\rangle$ and $\langle a, c\rangle$.
- `duplicates`: a duplicate structure, i.e., an activity appears in two positions in the process. This model allows only for a single trace variant $\langle a, b, c, b, d\rangle$. Modeling this structure is challenging because it requires two b transitions to restrict the number of occurrences of b - and most discovery approaches use a one-to-one mapping between activities and transitions.
- `non-free-choice`: a choice in a later part of the process depends on an earlier choice. Traces generated by this model start either with a or b (i.e., a free choice between the activities a and b). a is followed by a simple parallel structure between c and d. Similarly, b is followed by a simple parallel structure between e and d. Finally, there is a non-free choice between f and g. Traces with a and c end with f, and traces with b and e end with g.

Moreover, for each of the four structures, the corresponding base model is used to generate 30 event logs after adding different levels of noise. An infrequent activity x is used to add noise, and the following levels of noise are created:

- `no-noise`: activity x does not appear in the event log;

- `little-local`: activity x very rarely (in less than 5% of the traces) appears at a particular place in the process;
- `local`: activity x sometimes (in about 5% - 15% of the traces) appears at a particular place in the process;
- `semi-local`: activity x sometimes (in about 5% - 15% of the traces) appears at one of two places in the process;
- `little-global`: activity x very rarely (in less than 5% of the traces) appears at some random place in the process;
- `global`: activity x sometimes (in about 5% - 15% of the traces) appears at some random place in the process.

For each combination of structure and noise level, our data set contains five example logs. Each event log records $1\,000$ process instances. The logs are labeled by combining the base structure (`parallel`, `skip`, `duplicates`, or `non-free-choice`), the level of noise, and an additional letter (a, b, c, d, or e) to distinguish between the five example logs for each combination of base structure and noise level. For example, `non-free-choice_little-global_b` is the second example log (composed of $1\,000$ traces) that records the execution of the `non-free-choice` process with a very low frequency of global noise.

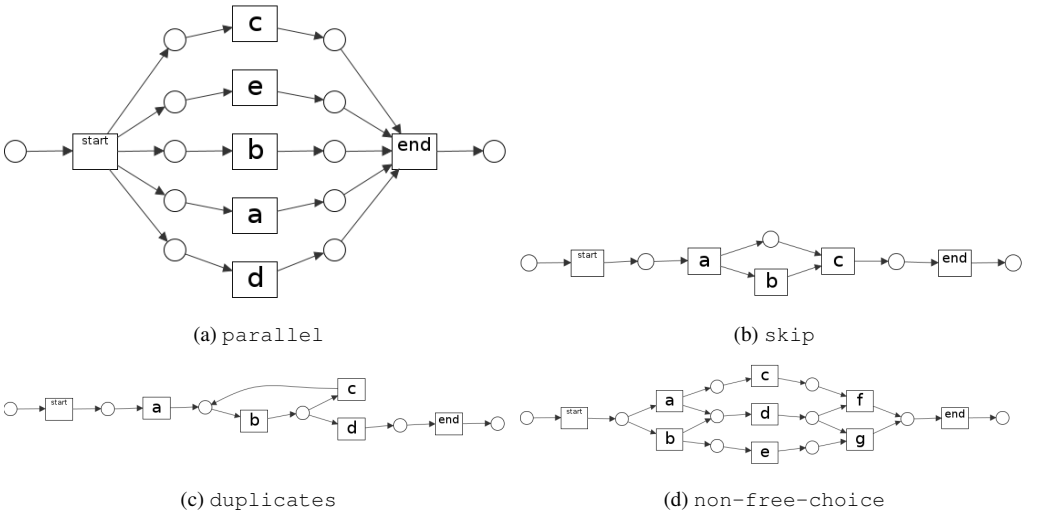
For the representational bias evaluation, we compare the performance of the two versions of the hybrid miner (HM_OV and HM_NV) for all the 120 event logs. We use as parameters $t_{freq} = 0$, $w = 0.5$, $t_{R_S} = 0.3$, $t_{R_W} = 0.2$ and $t_{replay} = 1.0$. We set low values for t_{R_S} and t_{R_W} as we aim to discover as many relations as possible and we set the t_{replay} to 1 as we want to ensure a perfect fitness to focus on how precisely the algorithm is able to model challenging structures.

Since the representational bias logs are of small size, both versions of the Hybrid Miner were able to discover a model in less than 5 seconds for each log. Therefore, we omit the time performance results and present the qualitative results of the experiment. All discovered models achieved a fitness score of 1 because we used a place evaluation threshold of 1. The precision values, instead, range according to the specific construct of the model. Table 6 reports the different precision values for each of the five example logs (a, b, c, d, or e) related to a specific representational bias and level of noise. Since no difference in terms of distribution of the precision values is observed between HM_OV and HM_NV, we report them only once.

Base structure	Noise level	a	b	c	d	e
parallel	no-noise	1	1	1	1	1
	little-local	0.751	0.752	0.766	0.766	0.944
	local	0.958	0.945	0.953	0.769	0.763
	semi-local	0.785	0.795	0.792	0.780	0.792
	little-global	0.744	0.757	0.755	0.759	0.731
	global	0.749	0.767	0.771	0.771	0.770
skip	no-noise	0.550	0.550	0.550	0.550	0.550
	little-local	0.420	0.420	0.420	0.420	0.421
	local	0.570	0.421	0.422	0.569	0.423
	semi-local	0.483	0.487	0.483	0.482	0.486
	little-global	0.576	0.572	0.578	0.576	0.574
	global	0.575	0.577	0.574	0.576	0.573
duplicates	no-noise	0.778	0.778	0.778	0.778	0.778
	little-local	0.472	0.472	0.471	0.470	0.470
	local	0.472	0.472	0.472	0.473	0.472
	semi-local	0.530	0.528	0.532	0.530	0.530
	little-global	0.756	0.754	0.753	0.754	0.755
	global	0.751	0.757	0.751	0.753	0.741
non-free-choice	no-noise	1	1	1	1	1
	little-local	0.632	0.631	0.634	0.633	0.633
	local	0.995	0.996	0.993	0.995	0.631
	semi-local	0.632	0.634	0.633	0.631	0.635
	little-global	0.812	0.843	0.794	0.810	0.841
	global	0.850	0.849	0.851	0.848	0.849

Table 6: Precision results for the representational bias evaluation. For each type of representational bias and for each level of noise, precision is collected for each of the five example logs (a, b, c, d, or e).

As example results, we show the models discovered for the noise-free logs (no-noise) in Figure 21 and the models discovered for the first example log with global noise (global_1000a) in Figure 22. For the non-free-choice and parallel cases, optimal models are discovered regardless of the noise level. The Hybrid Miner achieved a precision of 1.0 for the noise-free parallel and non-free-choice logs (see Figure 21a and 21d), while for other levels of noise lower values are observed. For example, in the model shown in Figure 22a, arcs of *Type II* are used for connecting the noisy activity x to $start$ and end, as well as for enabling the x self-looping. In the model shown in Figure 22d, arcs of *Type II* allow for executing x after $start$ and before end and a, while



(a) `parallel` (b) `skip`

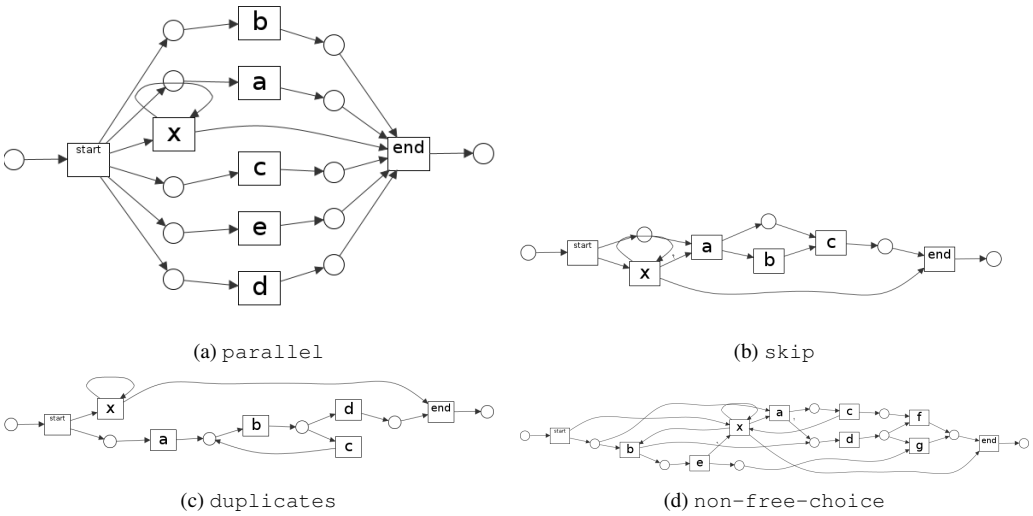
(c) `duplicates` (d) `non-free-choice`

Figure 21: Representational biases with no noise.

an unsure arc connects e to x. Although the base structures were correctly discovered for all `non-free-choice` and `parallel` log, the introduced noise led to lower precision values for the noisy logs. However, the decrease in precision is random since the noise was also inserted randomly. For example, for the `non-free-choice_1000a` log, the `local` log achieved the highest precision among the noisy logs (0.995), while for the `non-free-choice_1000e` log, the `global` log achieved the highest precision among the noisy logs (0.849).

Unlike the `parallel` and `non-free-choice` cases, the Hybrid Miner is not able to discover optimal models for the `skip` and `duplicates` cases. The noise-free `skip` logs led to a precision of 0.55 (Figure 21b). This is caused by the fact that b is continuously enabled considering only places and ignoring the sure arcs connected to b. The Hybrid Miner is not able to formally model a skip structure because it does not use silent transitions. A place was discovered to model the sequential relation between a and c, and the causal relations related to the optional activity b are modeled informally using two arcs of *Type II*. The models discovered for the `duplicates` logs are also non-optimal. For example, the noise-free `duplicates` logs led to a precision of 0.778 (Figure 21c). Since the Hybrid Miner does not allow for duplicate transitions, the structure is modeled through a loop that allows for repeating the sequence (c - b) arbitrarily often after the first execution of b.

By looking at the other `skip` and `duplicates` logs (`little-local`,



(a) `parallel` (b) `skip`

(c) `duplicates` (d) `non-free-choice`

Figure 22: Representational biases for the first example log with global noise (`global_1000a`).

`local`, `semi-local`, `little-global`, and `global`), the noise of the logs does not hamper the discovery of the same structures obtained with the `no-noise` logs, while enriching the models with transition x. For example, let us consider the models shown in Figure 22b and Figure 22c. In both models, arcs of *Type II* connect the noisy activity x to *start* and *end* and are used for a self-loop over x. Moreover, in the case of `skip`, an unsure arc also connects x to a. Although, as in the `no-noise` case, the Hybrid Miner is not able to correctly capture these two types of constructs, the introduction of the noise results in a slightly increased precision in some cases (e.g., from 0.55 to 0.572 for `skip_little-global_1000b`), and in a precision decrease in other cases (e.g., from 0.778 to 0.472 for `duplicates_little-local_1000a`). For all the discovered models, however, the noisy behavior is correctly represented as informal.

The results overall show that when "sure" behavior cannot be captured in formal constructs, it could make sense to capture the causalities in an informal way rather than ignoring them or trying to formalize them. For example, in Figure 21b, it is helpful to see that b is connected to a and c using sure arcs.

8.4. State-of-the-art discovery approaches

In this section, we compare HM_NV with one of the classical discovery approaches, the Inductive Miner (IM). We use two types of metrics for this comparison: (1) traditional fitness and precision that measure the compliance between

Log	SCR		SCP		Recall		Precision	
	IM	**HM_NV**	**IM**	**HM_NV**	**IM**	**HM_NV**	**IM**	**HM_NV**
BPI2011	0.48	**1**	0.667	**1**	0.277	**0.552**	**0.485**	0.111
BPI2012	0.567	**1**	0.895	**1**	0.76	**0.815**	**0.755**	0.317
BPI2014	0.353	**1**	0.4	**1**	0.725	**0.85**	0.623	**0.667**
BPI2015	0.604	**1**	0.763	**0.98**	0.474	**0.499**	0.757	**0.839**
BPI2016	0.964	**1**	1	1	**0.761**	0.757	0.047	**0.097**
BPI2017	0.299	**1**	0.793	**0.987**	0.851	**0.912**	**0.702**	0.486

Table 7: Evaluation results for the comparison of IM with HM_NV. SCR and SCP are causal-graph-based metrics, while we use recall and precision to refer to the traditional metrics (i.e., Petri-net-based metrics).

a Petri net and an event log and (2) causal-graph-based metrics that measure the compliance between two causal graphs (Definition 11 and Definition 14).

For computing traditional fitness and precision, we transform hybrid Petri nets into Petri nets by ignoring all arcs of *Type II* and *Type III*. Informal arcs are taken into account when computing causal-graph-based metrics. For causal-graph-based metrics, we transform the discovered Petri nets and Hybrid Petri nets into causal graphs[7], and we check the compliance between these causal graphs and the base causal graph (i.e., the causal graph discovered in the first discovery step of the hybrid miner). Note that this comparison is biased since the hybrid Petri net is constructed from the causal graph, i.e., we assume the base causal graph to be perfectly capturing causal dependencies between activities. Since a causal graph of a Petri net has no arcs of *Type III*, we only compute Strong Causal Recall (SCR) and Strong Causal Precision (SCP).

We use the same real-life event logs we used in Section 8.1 (i.e., the six BPI Challenge event logs), and we also use the same parameters for the HM_NV (i.e., the parameters reported in Table 2). For IM, we first filter the event logs to only keep the most frequent variants according to the parameter t_{freq}, then we apply IM with a noise filtering threshold of $(1 - t_{replay})$.

The results of this evaluation are reported in Table 7. HM_NV achieves better results with respect to the causal-graph-based metrics. For all event logs, HM_NV achieved a strong causal recall of 1 and a strong causal precision higher than or equal to 0.98. These results show that the second discovery step of HM_NV

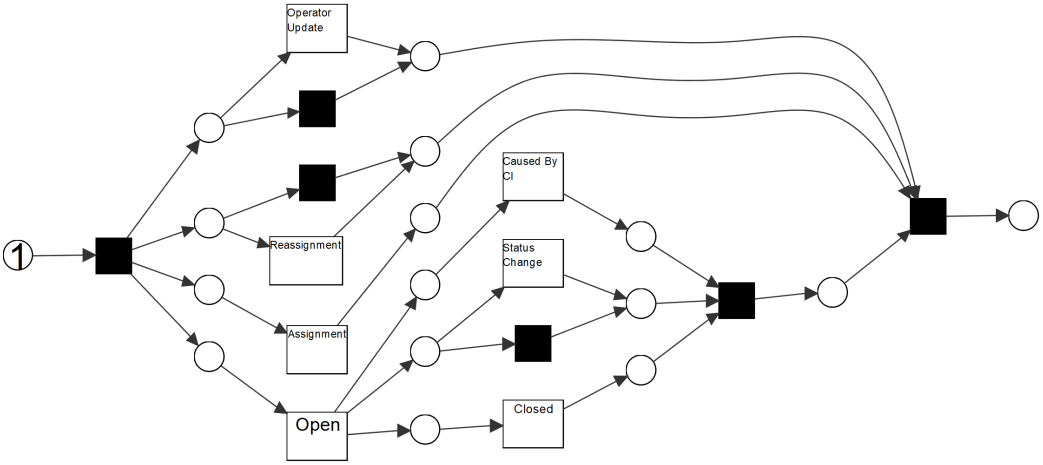
[7]Since the inductive miner produces models with silent transitions, we need to filter them out in the derived causal graph, i.e., two activities are considered to be causally related if there is a path connecting two transitions having the corresponding labels through silent transitions.

preserves the dependencies discovered in the first step (perfect SCR) and only a limited number of additional dependencies are added through the places (very high SCP). For IM, these values are lower except for the BPI2016, where both IM and HM_NV achieved a perfect strong causal precision. As mentioned before, these results are biased as the IM Petri net is discovered based on the event log, while the hybrid Petri net is discovered based on the base causal graph we used for computing these metrics.

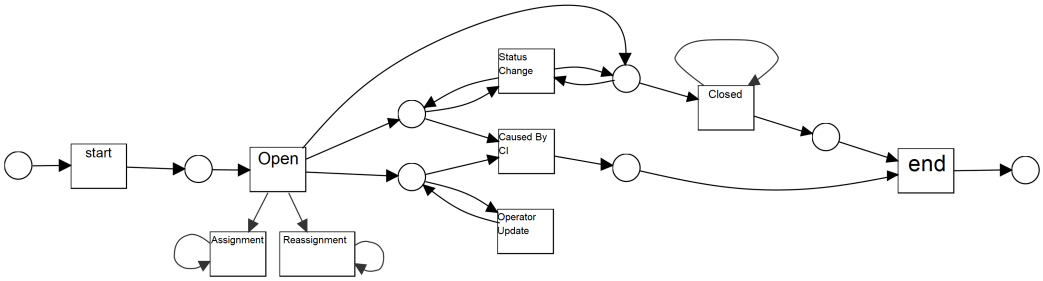
For the Petri net-based metrics (i.e., traditional fitness and precision), we observe different results among the event logs. For most logs, HM_NV led to models of higher recall. This was expected since the hybrid miner only adds places when there is enough evidence in the data justifying adding such formal constraints, while the IM models complex structures using silent transitions, loops, and concurrency structures. With respect to precision, we have cases where IM led to better results and other cases where HM_NV led to better results.

Figure 23 shows the models discovered for the BPI2014 event log. The model discovered by HM_NV is better in terms of all four quality metrics we use. The hybrid Petri net only allows for executing the activities *Status Change* and *Operator Update* before *Caused By CI*, while these three activities are concurrent in the Petri net discovered by IM. Moreover, in contrast to the Petri net discovered by IM, the Hybrid Petri net discovered by HM_NV allows for repeated executions of the activities *Status Change*, *Operator Update*, *Assignment*, and *Reassignment*.

Figure 24 shows the models discovered for the BPI2012 event logs. The hybrid Petri net discovered by HM_NV achieves a higher recall and a lower precision compared to the Petri net discovered by IM. The main reason for this drop in precision is the informal representation of self-loops in the hybrid Petri net. The hybrid Petri net contains self-loops of the activities *W_Complete aanvraag* and *W_Nabellen offertes*, and both activities are only connected to informal arcs, i.e., HM_NV was not able to formally model these self-loops because silent transitions are not supported in hybrid Petri nets. Since we omit informal arcs when computing precision, such activities that are not connected to any places yield a low precision. Overall, however, there is no clear winner between HM_NV and IM. Indeed, there are cases in which IM allows for getting the best trade-off between precision and recall (as in the case of BPI2012 and BPI2017), and cases in which HM_NV is able to get the best trade-off between precision and recall (e.g., BPI 2014 and BPI2015).

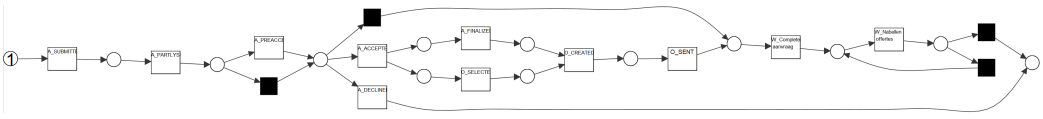


(a) Petri net discovered by IM.

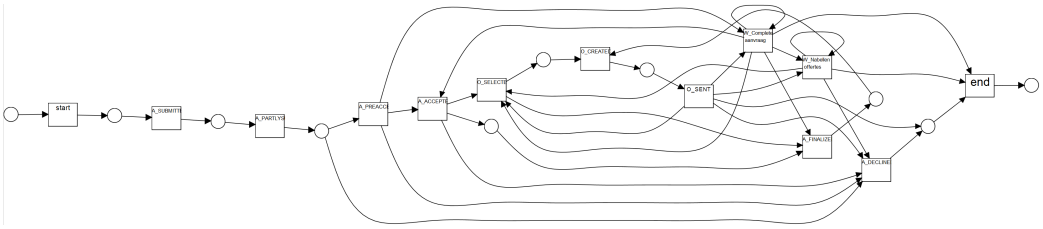


(b) Hybrid Petri net discovered by HM_NV.

Figure 23: Models discovered for the log BPI2014.



(a) Petri net discovered by IM.



(b) Hybrid Petri net discovered by HM_NV.

Figure 24: Models discovered for the log BPI2012.

8.5. Classification Task Using Process Discovery Contest Logs

Although fitness (also called recall) can be measured easily, there is less consensus on how to measure precision [42]. To evaluate a classifier and measure precision and recall directly, we need to have both positive and negative instances and a training and test log. In practical applications of process discovery, we only have positive examples and not a test log. The event log does not tell what is impossible in the process and one uses all the data available to get the best results. However, to evaluate a process discovery algorithm (i.e., not a process model), we can create synthetic data with positive and negative examples. This is the idea behind the annual *Process Discovery Contest*. In this contest both a training set and a separate test set with cases that are fitting and cases that are non-fitting are provided. This way process discovery can be treated as a classification problem, where a case is fitting or non-fitting.

We hence used the *BPI Process Discovery Contest 2016* (PDC2016) [43], the *BPI Process Discovery Contest 2017* (PDC2017) [44], and the *BPI Process Discovery Contest 2019* (PDC2019) [45] in order to evaluate the capability of the approach to correctly classify fitting and non-fitting traces, by computing precision, recall and F-measure. While precision measures the percentage of traces correctly classified as fitting with respect to all traces and recall the percentage of traces correctly classified as fitting with respect to all fitting traces, the F-measure is their harmonic mean. They all provide a measure of how accurate the model is in classifying traces as fitting or non-fitting. In detail, we compared the results obtained by the Hybrid Miner (both HM_OV and HM_NV) with the ones obtained by applying IM. Each of the three data sets is composed of 10 training and 10 test logs generated using ten BPMN models. Each training log contains 1,000 cases fitting the original model. The test logs, which are used for testing the quality of the discovered models, are composed of 20 traces, 10 traces fitting the model and 10 traces that do not fit the model used to generate the training log.

We used the following values for the parameters of both versions of the Hybrid Miner: $t_{freq} = 0, t_{R_S} = 0.5, t_{R_W} = 1.0, w = 0.5$. Moreover, in order to avoid overfitting of the training event logs, we decided to use a place evaluation threshold of 0.9 ($t_{replay} = 0.9$). For IM, we used the plug-in *Mine Petri net with Inductive Miner* from the ProM framework [46, 47]. Since we used a place evaluation threshold of 0.9 for the Hybrid Miner, we set the noise threshold of the *Inductive Miner* to 0.1.

Both the Hybrid Miner versions and the IM were able to discover a model in a few seconds for each log. Therefore, we omit the time performance results and focus on the qualitative analysis of the discovered models.

Event log	F-measure			Accuracy		
	IM	HM_OV	HM_NV	IM	HM_OV	HM_NV
PDC2016_1	0.947	**1**	**1**	0.95	**1**	**1**
PDC2016_1	0.947	**1**	**1**	0.95	**1**	**1**
PDC2016_2	0.947	**0.952**	**0.952**	0.95	0.95	0.95
PDC2016_3	0.462	0.462	**0.571**	0.65	0.65	**0.7**
PDC2016_4	0.182	0.182	0.1818	0.55	0.55	0.55
PDC2016_5	0.667	**1**	**1**	0.75	**1**	**1**
PDC2016_6	0.556	**0.741**	**0.741**	0.6	**0.65**	**0.65**
PDC2016_7	0.947	**0.952**	**0.952**	0.95	0.95	0.95
PDC2016_8	0.818	**0.833**	**0.833**	0.8	0.8	0.8
PDC2016_9	**1**	**1**	0.947	**1**	**1**	0.95
PDC2016_10	0.462	**0.889**	**0.889**	0.65	**0.9**	**0.9**
PDC2017_1	0.8	**0.87**	0.833	0.75	**0.85**	0.8
PDC2017_2	0.952	0.952	0.952	0.95	0.95	0.95
PDC2017_3	0.778	0.824	**0.947**	0.8	0.85	**0.95**
PDC2017_4	0.818	**0.909**	**0.909**	0.8	**0.9**	**0.9**
PDC2017_5	0.769	**0.833**	**0.833**	0.7	**0.8**	**0.8**
PDC2017_6	0.947	**0.952**	**0.952**	0.95	0.95	0.95
PDC2017_7	0.64	**0.667**	**0.667**	**0.55**	0.5	0.5
PDC2017_8	**0.87**	0.8	0.8	**0.85**	0.75	0.75
PDC2017_9	0.625	**0.8**	**0.8**	0.7	**0.8**	**0.8**
PDC2017_10	0.632	**0.769**	**0.769**	0.65	**0.7**	**0.7**
PDC2019_1	0.786	**1**	**1**	0.8	**1**	**1**
PDC2019_2	**1**	0.916	**1**	**1**	0.922	**1**
PDC2019_3	0.753	0.932	**0.978**	0.7889	0.933	**0.978**
PDC2019_4	0.703	0.936	**0.96**	0.756	0.933	**0.956**
PDC2019_5	**0.957**	0.884	0.884	**0.957**	0.878	0.878
PDC2019_6	0.989	**0.989**	**0.989**	0.989	0.989	0.989
PDC2019_7	0.782	**0.849**	**0.849**	0.789	**0.822**	**0.822**
PDC2019_8	0.605	**0.901**	**0.901**	0.478	**0.9**	**0.9**
PDC2019_9	0.541	**0.909**	**0.909**	0.622	**0.9**	**0.9**
PDC2019_10	0.547	**0.923**	**0.923**	0.411	**0.922**	**0.922**
average	0.749	0.854	**0.864**	0.771	0.857	**0.863**

Table 8: Results obtained by applying the Inductive Miner and the two versions of the Hybrid Miner to the BPI Process Mining Contest data sets.

Table 8 reports the F-Measure and accuracy results obtained by IM, HM_NV and HM_OV on the PDC2016, PDC2017 and PDC2019 data sets. The highest F-measure and accuracy values are highlighted in bold. By looking at the table we can observe that, for most cases, both versions of the Hybrid Miner were able to discover models of either a higher quality than the models discovered by IM or of the same quality. For instance, let us consider the models shown in Figure 25,

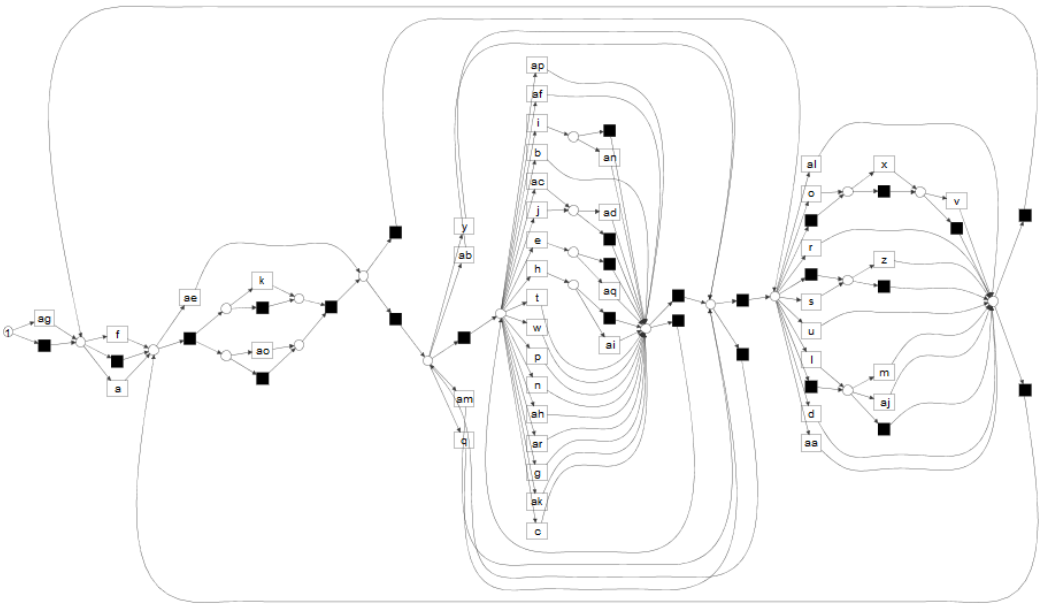
which were discovered by IM and HM_NV for the log PDC2019_8. Since the Inductive Miner does not support duplicate transitions, it tries to formally capture the whole process by adding many loops and silent transitions. This leads to a model of low quality compared to the model discovered by HM_NV. However, in the later model parts of the process are not formally captured. The model discovered by IM achieves an accuracy of 0.478 compared to 0.9 for the models discovered by the two versions of the Hybrid Miner. This example demonstrates how trying to formally capture vague behavior can decrease the quality of the process model. On the other hand, we observe some cases where IM delivered better results than the Hybrid Miner. Figure 26 shows the models discovered for the log PDC2017_8. The model discovered by IM has an accuracy of 0.85 compared to 0.75 achieved by the models discovered by the Hybrid Miner. IM, indeed, is able to formally capture more parts of the process through many additional places. These additional places cannot be generated by the Hybrid Miner because it does not support the usage of silent transitions.

For most logs, the two versions of the Hybrid Miner produce models having the same quality. There are a few exceptions with small differences in the quality scores. For instance, the model discovered by HM_OV for PDC2017_1 has an accuracy of 0.85 compared to 0.8 obtained for the model discovered by HM_NV. On the other hand, HM_NV achieved better results for PDC2017_3 (an accuracy of 0.85 for HM_OV and 0.95 for HM_NV). We can say, in general, that both versions of the Hybrid Miner delivered similar results in terms of quality, and these results are typically better than the results obtained by the Inductive Miner. Considering all process discovery contest logs, the models discovered by the Inductive Miner achieved an average accuracy of 0.77 and an average F-measure of 0.75. The models discovered by the Hybrid Miner achieved higher values: An average accuracy of 0.86 and an average F-measure of 0.86, with only minor differences between the two versions.

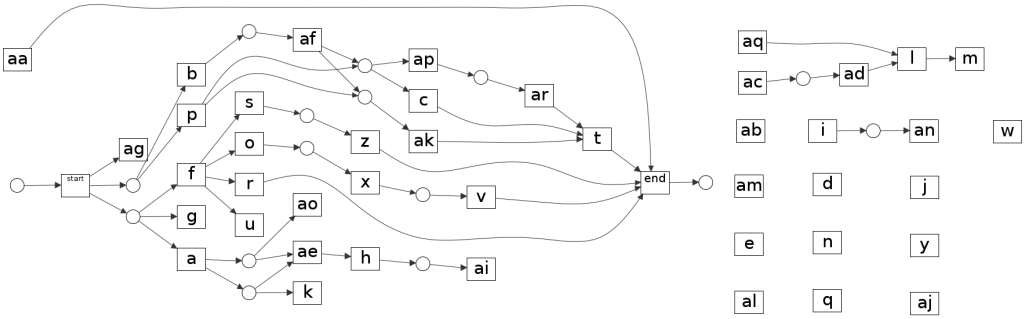
8.6. Parameter Evaluation

In this section, we evaluate the effect of changing the values of the different parameters of HM_NV on time performance and on the quality of the discovered models. We investigate the following parameters:

- the activity filter threshold t_{freq},
- the strong causality threshold t_{R_S},
- the causality weight w, and



(a) Petri net discovered by IM.

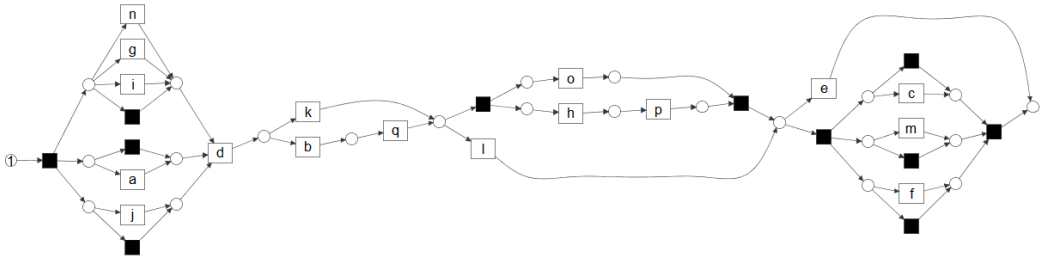


(b) Hybrid Petri net discovered by HM_NV.

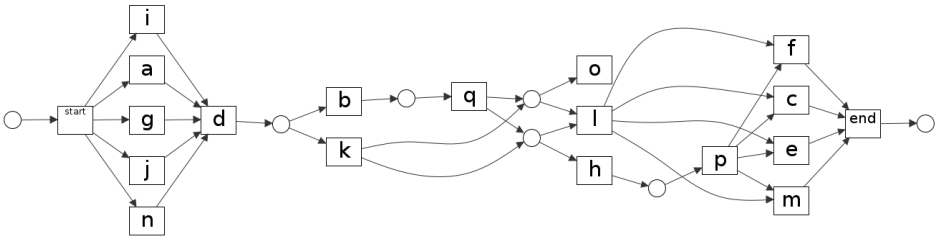
Figure 25: Models discovered for the log PDC_2019_8.

- the place quality evaluation threshold t_{replay}.

We omit the analysis related to the weak causality threshold (t_{R_W}) since uncertain edges are not used for building places and their effect on the quality of the discovered models cannot be measured using formal metrics. For each experiment presented in this section, we fixed all parameters except the parameter(s) tested by the experiment. We used the following values for the fixed parameters: $t_{freq} = 0$, $t_{R_S} = 0.5$, $t_{R_W} = 1$, $w = 0.5$, $t_{glob} = 0$, $t_{replay} = 0.8$. For all experiments, we used four event logs from the scalability data sets described in Section 8.2. We



(a) Petri net discovered by the Inductive Miner.



(b) Hybrid Petri net discovered by HM_NV.

Figure 26: Models discovered for the log PDC_2017_8.

selected one log to represent each of the four structures covered by the scalability data sets: PARALLEL_2_10000, SEQUENCE_2_10000, CHOICE_2_10000, and LOOP_2_10000. Each of these four logs contains 10 000 traces and its underlying model consists of two base components.

Activity Filter Threshold (t_{freq}).

We evaluated the activity filtering threshold by testing the following values: 0, 2 000, 4 000, 6 000, and 8 000. Figure 27 shows the performance results for this experiment (considering speed). As expected, the time costs decrease by using higher values for t_{freq}. Increasing the value of the activity filtering threshold reduces the number of nodes in the causal graph, and this, in turn, reduces the number of candidate places that need to be evaluated.

Figure 28 shows the qualitative results of the evaluation of the activity filtering parameter. For each log, fitness and precision scores of the discovered models are presented. We observe that increasing the value of t_{freq} reduces the fitness of the discovered models. This result was expected because using higher values for the activity filtering threshold reduces the number of transitions in the final models, and all traces containing any filtered activity become non-fitting. A clear trend of how the precision of the models changes by changing the value of t_{freq} is not easy

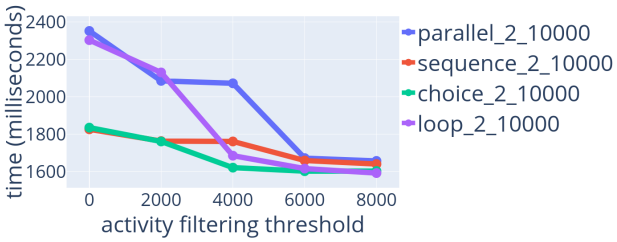
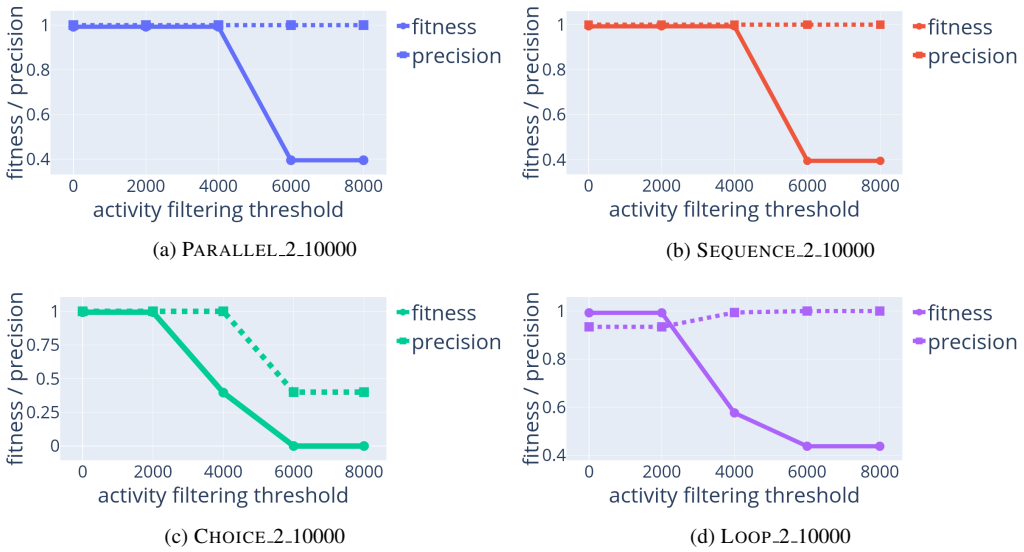


Figure 27: Time performance results of the evaluation of the parameter t_{freq}.



(a) PARALLEL_2_10000

(b) SEQUENCE_2_10000

(c) CHOICE_2_10000

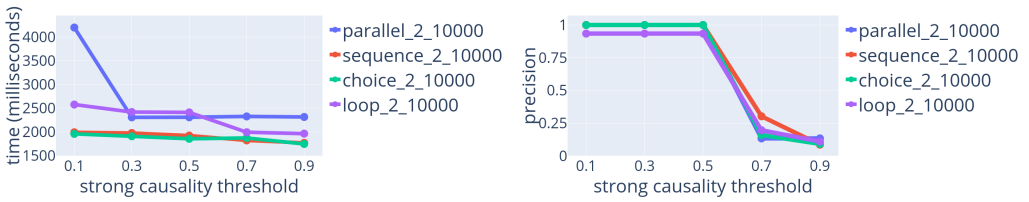
(d) LOOP_2_10000

Figure 28: Qualitative results of the evaluation of the parameter t_{freq}. Increasing the threshold leads to the removal of infrequent activities and hence lower fitness.

to identify for these event logs by looking at the plots.

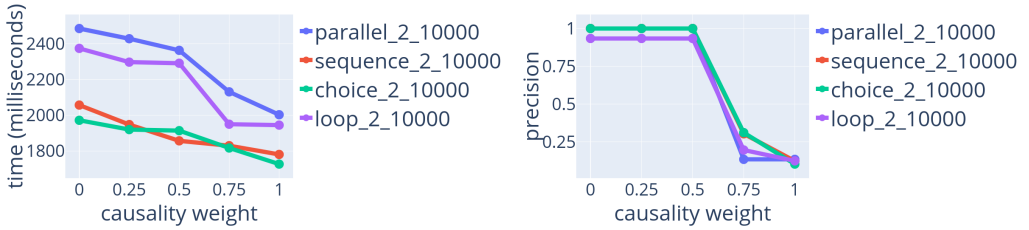
Strong Causality Threshold (t_{R_S}).

We evaluated the threshold for strong causalities by testing the following values: 0.1, 0.3, 0.5, 0.7, and 0.9. Figure 29 shows the results of this experiment. It demonstrates how, by decreasing the value of t_{R_S}, more precise models can be discovered, but more time is required. These results were expected because lowering the threshold for strong causalities will increase the number of sure edges in the causal graph, and this will create more candidate places to evaluate. Although we can generally state that a lower strong causality threshold leads to more precise models, we observe that for all four logs, the optimal model (the model with the



(a) Time (b) Precision

Figure 29: Results of the evaluation of parameter t_{R_S}. Fitness values are not reported because all discovered models achieved a fitness higher than 0.99.



(a) Time (b) Precision

Figure 30: Results of the evaluation of parameter w. Fitness values are not reported because all discovered models achieved a fitness higher than 0.99.

highest precision) was already discovered using a threshold of 0.5.

Causality Weight ($\boldsymbol{w}$).

For evaluating the effect of changing the causality weight parameter (w), we used the following values: 0, 0.25, 0.5, 0.75, and 1. Figure 30 shows the results of this evaluation. We observe that decreasing the value of w leads to more precise models, but increases the required time. Using a lower value for the causality weight w means giving more weight to loop and parallelism detection and less weight to split and join behavior. The results show hence that $Rel2$ is more important than $Rel1$ for the quality of the discovered models on these event logs. By investigating the discovered causal graphs, we can notice that using lower values for w increases the number of discovered causal edges and hence of candidate places. This explains the increase in both time and precision. For these event logs, it seems that the optimal model (the model with the highest precision) is already discovered using a causality weight of 0.5, at the lowest time performance cost.

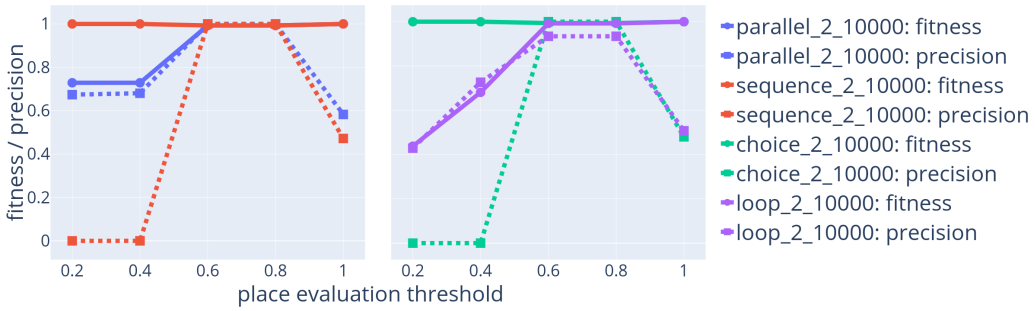


Figure 31: Qualitative results of the evaluation of parameter t_{replay}.

Place Quality Evaluation Threshold (t_{replay}).

We evaluated the threshold for place quality by testing the following values: 0.2, 0.4, 0.6, 0.8, and 1. Figure 31 shows the qualitative results of this experiment. It demonstrates how by increasing the value of t_{replay} the quality of the discovered models is improved in terms of both fitness and precision scores. However, we observe a large drop in precision when reaching the value $t_{replay} = 1$. This behavior can be justified by the fact that our four logs were not filtered to remove noise; using a threshold of 1 will prevent adding any places that are not perfectly fitting all noisy traces, and this will lower the precision of the model.

The time performance results of the experiment for the place quality threshold are reported in Table 9. No big differences are identified for the executions carried out with different values of t_{replay} and a clear trend of how the time performance of the program changes by changing the value of t_{replay} cannot be clearly identified. This behavior was expected because, although using a higher place evaluation threshold should speed up the rejection of non-fitting places, using a lower threshold might lead to accepting more places of maximal size and reducing the number of candidate places that need to be evaluated (see Section 6.3). This suggests that a good option is using high values for the place quality threshold (t_{replay}). However, when the data set is not noise-free, avoiding very high values seems to be the most suitable choice.

8.7. Discussion and Limitations

The in-depth analysis carried out allows us to answer the research questions posed earlier. By looking at the quantitative, qualitative, and performance-related comparison between HM_OV and HM_NV on real-life logs (Section 8.1), scalability logs (Section 8.2), and the logs used to evaluate different process constructs

t_{replay}	Time (milliseconds)			
	PARALLEL_2_10000	SEQUENCE_2_10000	CHOICE_2_10000	LOOP_2_10000
0.2	2 562	1 865	1 836	2 439
0.4	2 243	1 869	1 834	2 426
0.6	2 569	1 877	1 832	2 388
0.8	2 325	1 841	1 806	2 383
1	2 400	1 893	1 874	2 229

Table 9: Time performance results of the evaluation of parameter t_{replay}.

(Section 8.3), we can state that, overall no big differences in terms of model quality exist between HM_OV and HM_NV. However, there are important performance differences, especially for large event logs (**RQ1**). There are cases in which small differences in terms of fitness and precision can be observed. However, in general, HM_NV seems to return models that are of slightly better quality while requiring a fraction of compute time.

The results related to the comparison of the Hybrid Miner with the Inductive Miner on traditional Petri net-based and on causal-graph-based precision and recall - reported in Section 8.4 suggest us that the Hybrid Miner performs better than the Inductive Miner almost always in terms of Petri net-based recall and in half of the cases in terms of Petri net-based precision, while it performs always better in case of causal graph-based metrics. Moreover, the comparison of the approaches on a classification task - reported in Section 8.5 - allows us to assess that, overall, the two versions of the Hybrid Miner outperform the Inductive Miner on classification problems, while no big differences exist between HM_OV and HM_NV with slightly better results obtained with HM_NV (**RQ2**).

Overall, we can state that the parameter values have an impact on both the time performance and the quality of the discovered models (**RQ3**). Specifically, the higher the activity filtering threshold (t_{freq}), the lower the number of activities and the lower the fitness and the time required for discovering the models. We observe a similar result for the strong causality threshold (t_{R_S}). Also in this case, indeed, the higher the threshold, the lower the precision, as well as the time required for discovering the model. For the causality weight (w), we can also observe that the higher the value, the lower the precision - as it seems that *Rel2* is more important than *Rel1* at least for these logs – and the required time. Finally, we showed that the place quality evaluation threshold (t_{replay}) has an effect on precision and performance. Precision and time performance first improve when t_{replay} increases, but an important drop in precision occurs, as well as an increase of the required

time, when t_{replay} approaches 1 and the log includes noisy behavior.

The evaluation reveals that the Hybrid Miner (and in particular HM_NV) is able to discover reasonably good hybrid Petri nets that allow for formally representing the "sure" behavior and informally capturing noisy and infrequent behavior. This results in (1) fair fitness and precision values for these models - despite classical process mining metrics are not the most suitable ones for evaluating hybrid Petri nets; as well as (2) accurate classifications when the discovered hybrid Petri nets are evaluated on classification tasks. The time required for the computation is also acceptable. Except for complex and very large logs, the time required by HM_NV to discover a hybrid Petri net is of the order of seconds or a few minutes.

Despite the broad experimentation carried out, some limitations and threats related to the evaluation can be identified. The main threats affecting the validity of the evaluation carried out are related to external validity, limiting the generalizability of the results. A first threat is related to the fact that in the investigation fixed values have been selected for some of the Hybrid Miner parameters. However, the choices related to these values have always been well thought-out and motivated. A second threat is related to **RQ3**: the analysis of the impact of the parameter values on the discovered model is limited to a few event logs. This threat has been mitigated by the fact that investigated event logs are characterized by different base structures. We plan to carry on wider experimentation with more event logs in order to provide users with suggestions on the choice of the parameter values more suitable to their event logs.

9. Related Work

The work reported in this paper was inspired by the work of Herrmann et al. [48, 49] who argue that modeling "requires the representation of those parts of knowledge which cannot be stated definitely and have to be modeled vaguely". They propose annotations to make vagueness explicit. In [48, 49] the goal is to *model* vagueness, but we aim to automatically *discover* hybrid models supporting both vagueness and formal semantics.

Hybrid process models are related to the *partial models* considered in software engineering [50, 51]. In both papers, partial models represent sets of "normal" models. In Famelis et al. [50] one can use a so-called "may formula" which expresses allowable combinations of atoms. In Salay et al. [51] it is written "A partial model P consists of a base model, denoted $bs(P)$, and a set of annotations. Let T be the metamodel of $bs(P)$. Then, $[P]$ denotes the (possibly infinite) set

of models of type T called the concretizations of P. P is called consistent iff it allows at least one concretization, that is, $[P] \neq \emptyset$." Partial models in the spirit of [50, 51] are not specific for process modeling and do not address the topic of discovery. More related is the work on *configurable models* and *business process variability modeling*. See La Rosa et al. [52] for a recent survey. Most of the approaches allowing for variability and configuration extend a conventional process modeling language with constructs to capture customizable process models. A customizable process model represents a family of process variants in a way that a model of each variant can be derived by adding or deleting fragments according to customization options or according to a domain model [52]. There have been a few approaches for learning such models [53, 54, 55]. These either start from existing variant models as in Chen et al. [55] or from collections of event data, as in Buijs et al. and Gottschalk et al. [53, 54]. Partial and configurable models are different from the hybrid models considered here. We do not aim at a concretization of a hybrid model: The process is and will be "vague". Moreover, our input is not a *collection* of models [55] or logs [53, 54]. There is just a single event log.

Existing process discovery approaches can be split into approaches that produce *informal models* ("boxes and arrows") or *formal models* (Petri nets, transition systems, BPMN models, etc.).

Until now most of the commercial tools resort to discovering informal models, often based on the directly-follows graph with frequency-based filtering. There are over 40 commercial tools supporting process mining. Examples include: *Celonis Execution Management System* (Celonis), *ProcessGold Process Mining* (UiPath), *QPR ProcessAnalyzer* (QPR), *Disco* (Fluxicon), *minit* (Microsoft), *myInvenio* (IBM), *LanaLabs* (Appian), etc. These tools have been inspired by the first analysis step of the heuristic miner [7] (dependency graph) or the fuzzy miner [8] (highly configurable, but not allowing for any form of formal reasoning). Some of the commercial tools also support the discovery of BPMN models (e.g., Celonis support the Inductive Mining approach), but not as the first option.

In literature, one can find a range of process discovery approaches that produce formal models [1]. The α-algorithm in van der Aalst et al. [9] and its variants produce a Petri net. Approaches based on state-based regions [10, 11, 12, 13, 13], language-based regions [14, 15, 16, 17, 18, 19], abstract interpretation [56], Hasse diagrams [57], and inductive logic programming [58] also discover Petri nets. There are also techniques that learn causal nets [59, 7] or subsets of BPMN [60]. The more recently developed inductive mining approaches produce process trees that can be easily converted to Petri nets or similar [26, 27, 28].

The above approaches all discover a procedural model. Recently, also sev-

eral approaches have been proposed to learn declarative models [20, 21, 22, 23, 24, 25]. Often they use variants of the Declare language. The motivation for mining declarative models is that many of the formal algorithms tend to produce spaghetti-like process models or return flower-like models that impose no constraints. Declarative models tackle this by using an open-world assumption in combination with logic-based constraints: Anything is allowed unless explicitly forbidden by some constraint (expressed in terms of some logic or regular expressions). In this sense, Petri nets also use an open-world assumption and can therefore be considered as declarative. A Petri net without any places and just transitions allows for any behavior involving the activities represented by the transitions. Each place can be viewed as a constraint. Most discovery techniques do not exploit the declarative nature of Petri nets (tokens are assumed to trigger the next activity in a procedural manner). An exception is the work of Mannel et al. [61, 62] which improves discovery techniques based on regions by allowing for infrequent behavior. The approach is fully declarative in the sense that places are seen as constraints and not as a triggering mechanism. The concepts "declarative" and "hybrid" can be considered as orthogonal. Declarative process models could also be made hybrid by only expressing the "stronger" constraints (e.g., based on support, confidence or other measures such as the interestingness of a constraint [63]). and use other semi-formal means to show the causalities not covered by constraints.

Finally, recently, approaches combining procedural and declarative approaches have been proposed in the literature [64, 65, 66, 67]. The resulting models are also called "hybrid models", but the term "hybrid" is used differently. These combine procedural and declarative notations which are both formal, instead of adding semi-formal concepts like in this paper. The two notions of hybrid are orthogonal.

It is impossible to give a complete overview of all discovery approaches here. The initial approach presented in [29] was the first to return hybrid models having both formal and informal elements. This paper significantly improves and extends [29]. As described, the core algorithm (see Algorithm 1) has been changed in several ways:

- We now order the candidate places and have added the exclusion relation.
- The original approach did not use the conflict notion and this could result in contradicting places.
- We can now control the set of candidate places and put constraints on the connectivity of places.

- The approach has been reimplemented and now incorporates the newly developed speed-ups described in Section 6.3. Next to prioritizing the evaluation of highly connected candidate places and limiting the set of candidate places, we also stop the place evaluation as soon as possible and in a safe manner. Using the global place score we can further reduce the search space.

Moreover, all evaluations are new (Section 8) and we repositioned the approach based on experiences with more data sets (Section 2).

10. Conclusion

Hybrid models combine the best of two worlds: commercial tools producing informal models and more research-oriented discovery approaches providing formal guarantees. We provided a concrete realization of our hybrid discovery approach using *hybrid Petri nets*. The ideas are not limited to Petri nets and could be applied to other types of process models (e.g., BPMN models with explicit gateways for the clear and dominant behavior and additional arcs to capture complex or less dominant behavior). Unlike existing approaches, there is no need to straightjacket behavior into a formal model that suggests a level of confidence that is not justified. The places in the hybrid Petri nets have to satisfy strict quality criteria that can be interpreted by end-users. This is different from existing approaches where the models may not allow for any of the traces seen.

As for the limitations, our approach is not able to discover certain structures due to the absence of silent and duplicate transitions. In the future, we plan to extend it in order to deal with these constructs.

Our technique has been fully implemented in ProM and tested on numerous real-life event logs. Moreover, we reported on several controlled experiments showing that the approach indeed meets the requirements.

The explicit representation of vagueness and uncertainty in hybrid process models is analogous to the use of confidence intervals and box-and-whisker diagrams in descriptive statistics. This paper provides a starting point for a new branch of research in Business Process Management and process mining. Future work will include instantiations of the approach for BPMN and UML activity diagrams focusing on different model constructs (gateways, swimlanes, artifacts, etc.). The same ideas could also be used for very different process models (e.g., declarative process models and process trees). We would like to also provide approximative compliance and performance indicators for sure and unsure arcs.

Note that commercial tools show delays and frequencies on arcs, but these indicators may be very misleading as demonstrated in Figure 1.

Our broad evaluation focused on the choice of the most suitable Hybrid Miner parameters according to the characteristics of the log. Based on the intended purpose of the discovered model we aim to provide users with automated parameter-setting support when using Hybrid Miner plug-ins.

Future work is needed to make the approach even better scalable. The improvements described in Section 6.3 make the approach tractable in real-life settings. However, we would like to use integer linear programming and apriori-style approaches to tackle even larger event logs. Moreover, we are implementing the ideas in the context of big data processing engines like Apache Spark.

The approach in this paper also points towards a convergence of discovery and conformance checking. Commercial tools that support conformance checking (e.g., Celonis) use two representations: one for discovery (filtered Directly-follows Graphs) and one for conformance checking (e.g., a BPMN model). Clearly, this disconnect is undesirable and can only be solved by using hybrid process models. Moreover, we envision that in the future end-users will interact with discovery results and add/remove constraints (e.g., places) based on domain knowledge. Constraints may be normative and descriptive. Users can indicate that certain constraints are less interesting and others are essential and such information can be used to interactively show more meaningful conformance-checking results.

References

[1] W. van der Aalst, Process Mining: Data Science in Action, Springer-Verlag, Berlin, 2016.

[2] W. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer-Verlag, Berlin, 2011.

[3] W. van der Aalst, A. Adriansyah, B. van Dongen, Replaying History on Process Models for Conformance Checking and Performance Analysis, WIREs Data Mining and Knowledge Discovery 2 (2) (2012) 182–192.

[4] W. van der Aalst, M. Schonenberg, M. Song, Time Prediction Based on Process Mining, BPM Center Report BPM-09-04, BPMcenter.org (2009).

[5] W. van der Aalst, M. Pesic, M. Song, Beyond Process Mining: From the Past to Present and Future, in: B. Pernici (Ed.), Advanced Information Systems Engineering, Proceedings of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE'10), Vol. 6051 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2010, pp. 38–52.

[6] W. van der Aalst, Process Mining in the Large: A Tutorial, in: E. Zimanyi (Ed.), Business Intelligence (eBISS 2013), Vol. 172 of Lecture Notes in Business Information Processing, Springer-Verlag, Berlin, 2014, pp. 33–76.

[7] A. Weijters, W. van der Aalst, Rediscovering Workflow Models from Event-Based Data using Little Thumb, Integrated Computer-Aided Engineering 10 (2) (2003) 151–162.

[8] C. Günther, W. van der Aalst, Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics, in: G. Alonso, P. Dadam, M. Rosemann (Eds.), International Conference on Business Process Management (BPM 2007), Vol. 4714 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2007, pp. 328–343.

[9] W. van der Aalst, A. Weijters, L. Maruster, Workflow Mining: Discovering Process Models from Event Logs, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1128–1142.

[10] W. van der Aalst, V. Rubin, H. Verbeek, B. Dongen, E. Kindler, C. Günther, Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting, Software and Systems Modeling 9 (1) (2010) 87–111.

[11] J. Carmona, J. Cortadella, M. Kishinevsky, A Region-Based Algorithm for Discovering Petri Nets from Event Logs, in: Business Process Management (BPM 2008), 2008, pp. 358–373.

[12] J. Carmona, J. Cortadella, M. Kishinevsky, Genet: A Tool for the Synthesis and Mining of Petri Nets, in: Application of Concurrency to System Design (ACSD 2009), IEEE Computer Society, 2009, pp. 181–185.

[13] M. Solé, J. Carmona, Process Mining from a Basis of State Regions, in: Applications and Theory of Petri Nets (Petri Nets 2010), Vol. 6128 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2010, pp. 226–245.

[14] R. Bergenthum, J. Desel, R. Lorenz, S. Mauser, Process Mining Based on Regions of Languages, in: G. Alonso, P. Dadam, M. Rosemann (Eds.), International Conference on Business Process Management (BPM 2007), Vol. 4714 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2007, pp. 375–383.

[15] R. Bergenthum, J. Desel, R. Lorenz, S. Mauser, Synthesis of Petri Nets from Finite Partial Languages, Fundamenta Informaticae 88 (4) (2008) 437–468.

[16] R. Bergenthum, J. Desel, S. Mauser, R. Lorenz, Synthesis of Petri Nets from Term Based Representations of Infinite Partial Languages, Fundamenta Informaticae 95 (1) (2009) 187–217.

[17] R. Lorenz, J. Desel, G. Juhas, Models from Scenarios, in: K. Jensen, W. Aalst, G. Balbo, M. Koutny, K. Wolf (Eds.), Transactions on Petri Nets and Other Models of Concurrency (ToPNoC VII), Vol. 7480 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2013, pp. 314–371.

[18] J. van der Werf, B. van Dongen, C. Hurkens, A. Serebrenik, Process Discovery using Integer Linear Programming, Fundamenta Informaticae 94 (2010) 387–412.

[19] S. van Zelst, B. van Dongen, W. van der Aalst, Avoiding Over-Fitting in ILP-Based Process Discovery, in: H. Motahari-Nezhad, J. Recker, M. Weidlich (Eds.), International Conference on Business Process Management (BPM 2015), Vol. 9253 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2015, pp. 163–171.

[20] C. Di Ciccio, M. Mecella, A Two-Step Fast Algorithm for the Automated Discovery of Declarative Workflows, in: IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2013), IEEE Computer Society, 2013, pp. 135–142.

[21] C. Di Ciccio, F. Maggi, J. Mendling, Efficient discovery of Target-Branched Declare constraints, Information Systems 56 (2016) 258–283.

[22] C. Di Ciccio, M. Mecella, On the discovery of declarative control flows for artful processes, ACM Transactions on Management Information Systems 5 (4) (2015) 24:1–24:37.

[23] S. Ferilli, WoMan: Logic-Based Workflow Learning and Management, IEEE Transactions on Systems, Man, and Cybernetics: Systems 44 (6) (2014) 744–756.

[24] F. Maggi, R. Bose, W. van der Aalst, Efficient Discovery of Understandable Declarative Process Models from Event Logs, in: J. Ralyte, X. Franch, S. Brinkkemper, S. Wrycza (Eds.), International Conference on Advanced Information Systems Engineering (Caise 2012), Vol. 7328 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2012, pp. 270–285.

[25] S. Schönig, A. Rogge-Solti, C. Cabanillas, S. Jablonski, J. Mendling, Efficient and Customisable Declarative Process Mining with SQL, in: S. Nurcan, P. Soffer, M. Bajec, J. Eder (Eds.), International Conference on Advanced Information Systems Engineering (Caise 2016), Vol. 9694 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2016, pp. 290–305.

[26] S. Leemans, D. Fahland, W. van der Aalst, Discovering Block-structured Process Models from Event Logs: A Constructive Approach, in: J. Colom, J. Desel (Eds.), Applications and Theory of Petri Nets 2013, Vol. 7927 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2013, pp. 311–329.

[27] S. Leemans, D. Fahland, W. Aalst, Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour, in: N. Lohmann, M. Song, P. Wohed (Eds.), Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2013), Vol. 171 of Lecture Notes in Business Information Processing, Springer-Verlag, Berlin, 2014, pp. 66–78.

[28] S. Leemans, D. Fahland, W. van der Aalst, Scalable Process Discovery and Conformance Checking, Software and Systems Modeling 17 (2) (2018) 599–631. doi:10.1007/s10270-016-0545-x.

[29] W. van der Aalst, R. De Masellis, C. Di Francescomarino, C. Ghidini, Learning Hybrid Process Models From Events: Process Discovery Without Faking Confidence, in: J. Carmona, G. Engels, A. Kumar (Eds.), International Conference on Business Process Management (BPM 2017), Vol. 10445 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2017, pp. 59–76.

[30] J. Cortadella, M. Kishinevsky, L. Lavagno, A. Yakovlev, Deriving Petri Nets from Finite Transition Systems, IEEE Transactions on Computers 47 (8) (1998) 859–882.

[31] W. van der Aalst, On the Pareto Principle in Process Mining, Task Mining, and Robotic Process Automation, in: S. Hammoudi, C. Quix, J. Bernardino (Eds.), Proceedings of the 9th International Conference on Data Science, Technology and Applications, DATA 2020, Lieusaint, Paris, France, July 7-9, 2020, SciTePress, 2020, pp. 5–12.

[32] L. Cheng, B. van Dongen, W. van der Aalst, Scalable Discovery of Hybrid Process Models in a Cloud Computing Environment, IEEE Trans. Serv. Comput. 13 (2) (2020) 368–380. doi:10.1109/TSC.2019.2906203.
URL https://doi.org/10.1109/TSC.2019.2906203

[33] B. van Dongen, BPI Challenges (2011-2017), Real life Event Logs Collection, data.4tu.nl/repository/collection:event_logs (2017).

[34] B. van Dongen, Real-life event logs - Hospital log (4TU Data Set) (2011). doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.

[35] B. van Dongen, BPI Challenge 2012, dataset. http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f (2012).

[36] B. van Dongen, BPI Challenge 2017 (4TU Data Set) 10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b (2017). doi:10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b.

[37] B. van Dongen, BPI Challenge 2014 (4TU Data Set) 10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35 (2014). doi:10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35.

[38] B. van Dongen, BPI Challenge 2015 (4TU Data Set) 10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1 (2015). doi:10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1.

[39] M. Dees, B. van Dongen, BPI Challenge 2016: Clicks Logged In (4TU Data Set) 10.4121/uuid:

01345ac4-7d1d-426e-92b8-24933a079412 (2016). doi:10.4121/uuid:01345ac4-7d1d-426e-92b8-24933a079412.

[40] W. van der Aalst, Benchmarking Logs to Test Scalability of Process Discovery Algorithms (Event Data from 4TU.ResearchData), https://doi.org/10.4121/uuid:1cc41f8a-3557-499a-8b34-880c1251bd6e (2017).

[41] W. van der Aalst, Testing Representational Biases (Event Data from 4TU.ResearchData), url = https://data.4tu.nl/articles/dataset/Testing_Representational_Biases/12683159 (2017). doi:10.4121/uuid:25d6eef5-c427-42b5-ab38-5e512cca08a9.

[42] N. Tax, X. Lu, N. Sidorova, D. Fahland, W. van der Aalst, The Imprecisions of Precision Measures in Process Mining, Information Processing Letters 135 (2018) 1–8.

[43] J. Carmona, M. de Leoni, B. Depaire, T. Jouck, Process Discovery Contest 2016 (PDC2016), https://data.4tu.nl/articles/dataset/Process_Discovery_Contest_2016/14625912 (2016). doi:10.4121/14625912.v1.

[44] J. Carmona, M. de Leoni, B. Depaire, T. Jouck, Process Discovery Contest 2017 (PDC2017), https://data.4tu.nl/articles/dataset/Process_Discovery_Contest_2017/14625948 (2017). doi:10.4121/14625948.v1.

[45] J. Carmona, M. de Leoni, B. Depaire, T. Jouck, Process Discovery Contest 2019 (PDC2019), https://data.4tu.nl/articles/dataset/Process_Discovery_Contest_2019/14625996 (2019). doi:10.4121/14625996.v1.

[46] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters, W. van der Aalst, The ProM Framework: A New Era in Process Mining Tool Support, in: G. Ciardo, P. Darondeau (Eds.), Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings, Vol. 3536 of Lecture Notes in Computer Science, Springer, 2005, pp. 444–454. doi:10.1007/11494744_25.
URL https://doi.org/10.1007/11494744_25

[47] W. van der Aalst, B. van Dongen, C. Günther, A. Rozinat, E. Verbeek, T. Weijters, ProM: The Process Mining Toolkit, in: Proceedings of the Business Process Management Demonstration Track (BPMDemos 2009), Ulm, Germany, September 8, 2009, Vol. 489 of CEUR Workshop Proceedings, CEUR-WS.org, 2009.
URL http://ceur-ws.org/Vol-489/paper3.pdf

[48] T. Herrmann, M. Hoffmann, K. Loser, K. Moysich, Semistructured Models are Surprisingly Useful for User-Centered Design, in: G. Michelis, A. Giboin, L. Karsenty, R. Dieng (Eds.), Designing Cooperative Systems (Coop 2000), IOS Press, Amsterdam, 2000, pp. 159–174.

[49] T. Herrmann, K. Loser, Vagueness in Models of Socio-technical Systems, Behaviour and Information Technology 18 (5) (1999) 313–323.

[50] M. Famelis, R. Salay, M. Chechik, Partial Models: Towards Modeling and Reasoning with Uncertainty, in: International Conference on Software Engineering (ICSE 2012), IEEE Computer Society, 2012, pp. 573–583.

[51] R. Salay, M. Chechik, J. Horkoff, A. Sandro, Managing Requirements Uncertainty with Partial Models, Requirements Engineering 18 (2) (2013) 107–128.

[52] M. La Rosa, W. van der Aalst, M. Dumas, F. Milani, Business process variability modeling: A survey, ACM Computing Surveys 50 (1) (2017) 2:1–2:45.

[53] J. Buijs, B. van Dongen, W. van der Aalst, Mining Configurable Process Models from Collections of Event Logs, in: F. Daniel, J. Wang, B. Weber (Eds.), International Conference on Business Process Management (BPM 2013), Vol. 8094 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2013, pp. 33–48.

[54] F. Gottschalk, W. van der Aalst, M. Jansen-Vullers, Mining Reference Process Models and their Configurations, in: R. Meersman, Z. Tari, P. Herrero (Eds.), Proceedings of the 3rd International Workshop on Enterprise Integration, Interoperability and Networking, EI2N08, OTM 2008 Workshops, Vol. 5333 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2008, pp. 263–272.

[55] C. Li, M. Reichert, A. Wombacher, The MINADEPT Clustering Approach for Discovering Reference Process Models Out of Process Variants, International Journal of Cooperative Information Systems 19 (3-4) (2010) 159–203.

[56] J. Carmona, J. Cortadella, Process Mining Meets Abstract Interpretation, in: J. Balcazar (Ed.), ECML/PKDD 210, Vol. 6321 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, 2010, pp. 184–199.

[57] R. Bergenthum, Synthesizing Petri Nets from Hasse Diagrams, in: J. Carmona, G. Engels, A. Kumar (Eds.), International Conference on Business Process Management (BPM 2017), Vol. 10445 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2017, pp. 59–76.

[58] S. Goedertier, D. Martens, J. Vanthienen, B. Baesens, Robust Process Discovery with Artificial Negative Events, Journal of Machine Learning Research 10 (2009) 1305–1340.

[59] B. Vazquez-Barreiros, M. Mucientes, M. Lama, ProDiGen: Mining Complete, Precise and Minimal Structure Process Models with a Genetic Algorithm, Information Sciences 294 (Supplement C) (2015) 315 – 333.

[60] D. Redlich, T. Molka, W. Gilani, G. Blair, A. Rashid, Constructs Competition Miner: Process Control-Flow Discovery of BP-Domain Constructs, in: S. Sadiq, P. Soffer, H. Voelzer (Eds.), International Conference on Business Process Management (BPM 2014), Vol. 8659 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2014, pp. 134–150.

[61] L. Mannel, W. van der Aalst, Finding Complex Process-Structures by Exploiting the Token-Game, in: S. Donatelli, S. Haar (Eds.), Applications and Theory of Petri Nets 2019, Vol. 11522 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2019, pp. 258–278.

[62] L. Mannel, W. van der Aalst, Discovering Process Models with Long-Term Dependencies While Providing Guarantees and Handling Infrequent Behavior, in: L. Bernardinello, L. Petrucci (Eds.), Applications and Theory of Petri Nets 2022, Vol. 13288 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2022, pp. 303–324.

[63] A. Cecconi, G. De Giacomo, C. Di Ciccio, F. M. Maggi, J. Mendling, Measuring the interestingness of temporal logic behavioral specifications in pro-

cess mining, Inf. Syst. 107 (2022) 101920. doi:10.1016/j.is.2021.101920.
URL https://doi.org/10.1016/j.is.2021.101920

[64] M. Westergaard, T. Slaats, Mixing paradigms for more comprehensible models, in: F. Daniel, J. Wang, B. Weber (Eds.), Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings, Vol. 8094 of Lecture Notes in Computer Science, Springer, 2013, pp. 283–290. doi:10.1007/978-3-642-40176-3_24.
URL https://doi.org/10.1007/978-3-642-40176-3_24

[65] F. M. Maggi, T. Slaats, H. A. Reijers, The automated discovery of hybrid processes, in: S. W. Sadiq, P. Soffer, H. Völzer (Eds.), Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings, Vol. 8659 of Lecture Notes in Computer Science, Springer, 2014, pp. 392–399. doi:10.1007/978-3-319-10172-9_27.
URL https://doi.org/10.1007/978-3-319-10172-9_27

[66] D. M. M. Schunselaar, T. Slaats, F. M. Maggi, H. A. Reijers, W. M. P. van der Aalst, Mining hybrid business process models: A quest for better precision, in: W. Abramowicz, A. Paschke (Eds.), Business Information Systems - 21st International Conference, BIS 2018, Berlin, Germany, July 18-20, 2018, Proceedings, Vol. 320 of Lecture Notes in Business Information Processing, Springer, 2018, pp. 190–205. doi:10.1007/978-3-319-93931-5_14.
URL https://doi.org/10.1007/978-3-319-93931-5_14

[67] J. De Smedt, J. De Weerdt, J. Vanthienen, Fusion miner: Process discovery for mixed-paradigm models, Decis. Support Syst. 77 (2015) 123–136. doi:10.1016/j.dss.2015.06.002.
URL https://doi.org/10.1016/j.dss.2015.06.002