Data-driven Process Discovery - Revealing Conditional Infrequent Behavior from Event Logs

Felix Mannhardt[1], Massimiliano de Leoni[1] , Hajo A. Reijers[2,1],
Wil M.P. van der Aalst[1]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
[2] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
f.mannhardt, m.d.leoni, h.a.reijers, w.m.p.v.d.aalst@tue.nl

Abstract. Process discovery methods automatically infer process models from event logs. Often, event logs contain so-called noise, e.g., infrequent outliers or recording errors, which obscure the main behavior of the process. Existing methods filter this noise based on the frequency of event labels: infrequent paths and activities are excluded. However, infrequent behavior may reveal important insights into the process. Thus, not all infrequent behavior should be considered as noise. This paper proposes the Data-aware Heuristic Miner (DHM), a process discovery method that uses the data attributes to distinguish infrequent paths from random noise by using classification techniques. Data- and control-flow of the process are discovered together. We show that the DHM is, to some degree, robust against random noise and reveals data-driven decisions, which are filtered by other discovery methods. The DHM has been successfully tested on several real-life event logs, two of which we present in this paper.

Keywords: Process Mining, Process Discovery, Event Logs, Noise, Data Rules

1 Introduction

Process models are used by organizations to document, specify, and analyze their processes [1]. A process model describes the expected behavior of a process in terms of its activities (i.e., units of work) and their ordering. Most contemporary processes are supported by information systems. Often, those systems record information about the execution of processes in databases. With the abundance of such data, there is a growing interest in *process discovery* [2], i.e., revealing the actual execution of processes from events. Process discovery methods automatically infer process models from *event logs*.

One important challenge for process discovery methods is to handle event logs with *noise* [2,3]. In practice, event logs often contain noise, e.g., out-of-order events, exceptional behavior, or recording errors [4]. Including all such infrequent events in the process discovery often leads to unusable, complex models. Therefore, *noise filtering methods* that distinguish noise from the regular behavior of the process may be useful.

Some of the early techniques for process discovery assumed noise-free event logs (e.g., the Alpha algorithm [5] and the region based approaches [6]). These techniques are of limited use in real-life settings. Most of the more recent and more sophisticated process discovery methods support noise filtering [3]. Existing noise-filtering methods are based on frequencies [7,8,9,10], machine-learning techniques [11,12], genetic algorithms [13], or probabilistic models [14,15]. All of those methods focus on the *control-flow perspective* (i.e., the event labels) when filtering noise. Dedicated noise filtering methods [16,17] are also based on frequencies.

However, processes are often governed by rules. Decision are taken on the basis of available data, available resources, and the process context. Some paths may be executed infrequently because the corresponding conditions are rarely fulfilled. *Existing methods based solely on the control-flow perspective would disregard such infrequent behavior as noise.* However, some infrequent behavior may be characterized by very deterministic rules, and, thus, be of great interest to process analysts (e.g., in the context of risks and fraud). For example, shortcuts in a process might only be taken by a specific resource, undesired behavior might be subject to conditions, and infrequently actions might be legitimate only for special types of cases. These kind of events should not be set aside as *noise*. Methods exist to discover such decision rules [18,19,20] but all rely on a previously discovered process model of the process. Hence, existing methods do not leverage the full potential of the *data perspective*. Data- and control-flow need to be discovered together. Recent work on declarative process discovery [21] considers the data perspective. However, similar to association rule mining, sets of rules rather than full process models are returned.

In this work, we propose the Data-aware Heuristic Miner (DHM), which takes the data perspective into account when discovering the control flow of a process. The DHM uses classification techniques to reveal data dependencies between activities, and uses these data dependencies to *distinguish noise from infrequent conditional behavior*. It returns process models that yield a better insight into the data perspective of processes by revealing hidden data dependencies while filtering random noise. The evaluation on real-life cases shows that the DHM reveals additional insights *not* returned by state-of-the-art process discovery methods. We confirmed the discovered conditions with a domain expert for one of the real-life event logs. The experiment on the synthetic data shows that the DHM is resilient to a certain degree of randomly injected noise, which is not characterized by data conditions. It rediscovers the original model, whereas earlier techniques either show too much, or too little behavior. The *contribution of this paper* is a process discovery method that is able to *distill important information from infrequent behavior* instead of dismissing it as noise.

The remainder of this paper is structured as follows. We start by introducing the problem with an example in Sect. 2. Then, required preliminaries are introduced in Sect. 3. Section 4 presents our novel process discovery method. We evaluate our method using both synthetic and real-life data in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Problem Description

Figure 1 shows a simplified process from the health care domain. We use this example in the paper to motivate the relevance of the data perspective for noise filtering. When patients arrive at the hospital they are assigned a triage priority, registered and assigned to a responsible nurse. Only in exceptional cases, patients are assigned the *white* triage priority. Those patients typically, leave the emergency ward directly after registration

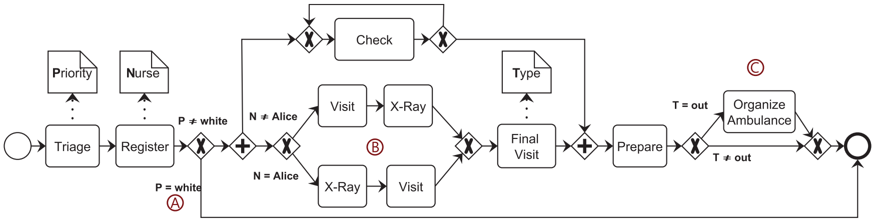
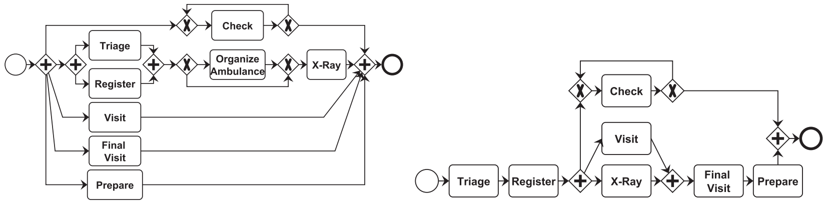


Fig. 1. A simplified process in BPMN notation from the emergency ward of a hospital, which is used as motivating example throughout this paper.



(a) IM filters little of the injected noise and fails to reveal behavior Ⓐ and Ⓑ.

(b) HM filters the injected noise well, but fails to show behavior Ⓐ, Ⓑ and Ⓒ.

Fig. 2. Models discovered by IM and HM on an event log generated from the example process.

since their injuries do not require the attendance of a doctor Ⓐ. All other patients are admitted to the emergency ward. While patients are in the emergency ward a nurse periodically checks their condition. In parallel to this, for the group of patients under consideration, an X-ray is taken and a doctor visits the patient. There are two different work practices regarding these two activities. Normally, the doctor visits the patients after which the X-Ray is taken. One particular nurse (Alice) re-sequences these activities in the reversed order to improve the process: first the X-ray is taken and, only thereafter, the doctor visits the patient Ⓑ. As this work practice is only followed by one nurse, it is observed less frequently. Afterwards, the doctor visits the patient one more time and decides on the type of dismissal. Then, the patient is prepared for a possible transfer. For patients with the *out* dismissal type an ambulance needs to be organized Ⓒ. This process contains three examples of infrequent, data-dependent behavior: Ⓐ a data-dependent path, Ⓑ data-dependent re-sequencing, and Ⓒ a data-dependent activity. The goal of our work is to rediscover such behavior, while ignoring random noise.

Assume an event log of the process in Figure 1 obtained from the information systems of the hospital. As motivated in the introduction, it is likely that this event log contains noise. We applied both the Heuristic Miner (HM) [8] and the Inductive Miner (IM) [9] as representatives of discovery methods supporting noise filtering on such an event log with a controlled degree of noise[1]. Figure 2 shows the resulting process mod-

[1] Here, in 5% of the cases one additional event was randomly executed out of the original order.

Table 1. Three traces of the example process with attributes **act**ivity, **p**riority, **n**urse, and **t**ype.

(a) Trace $\sigma_1 \in \mathscr{L}$

id	act	p	n	t
e_{11}	`Triage`	Red		
e_{12}	`Register`		Joe	
e_{13}	`Check`			
e_{14}	`Check`			
e_{15}	`Check`			
e_{16}	`Visit`			
e_{17}	`X-Ray`			
e_{18}	`F. Visit`			ICU
e_{19}	`Prepare`			

(b) Trace $\sigma_2 \in \mathscr{L}$

id	act	p	n	t
e_{21}	`Triage`	Red		
e_{22}	`Register`		Alice	
e_{23}	`Check`			
e_{24}	`X-Ray`			
e_{25}	`Visit`			
e_{26}	`Check`			
e_{27}	`F. Visit`			out
e_{28}	`Prepare`			
e_{29}	`Org. Amb.`			

(c) Trace $\sigma_3 \in \mathscr{L}$

id	act	p	n	t
e_{31}	`Triage`	Red		
e_{32}	`Register`		Joe	
e_{33}	`Check`			
e_{34}	`Visit`			
e_{35}	`X-Ray`			
e_{36}	`Check`			
e_{37}	`Check`			
e_{38}	`F. Visit`		NC	
e_{39}	`Prepare`			

els in BPMN notation. Clearly, both methods are unaware of the data perspective. Therefore, they fail to distinguish between random noise and the infrequent data-dependent behavior Ⓐ, Ⓑ, and Ⓒ. It might be possible to tweak the parameters of the algorithms such that more behavior is revealed (e.g., through grid search). Still, finding the correct parameter setting that does not include unrelated noise requires deep knowledge about the underlying process. Therefore, this is often not feasible. Moreover, it is not possible to reveal the infrequent data-dependent behavior by using decision mining techniques. Those techniques can only reveal decision rules for paths that are reflected in the process model, thus low-frequent but deterministic behavior remains undetected.

In the remainder of this paper, we describe the DHM, which extends the ideas of the HM with the use of classification techniques to reveal data dependencies. Our method, indeed, rediscovers the behavior of the process as shown in Figure 1.

3 Preliminaries

An **event log** stores information about activities that were recorded by information systems supporting the execution of a process [2]. Each execution of a *process instance* results in a sequence of events. Each events corresponds to the *execution of one activity*. Given universes of attributes A and values U, an event log $L = (E, \Sigma, \#, \mathscr{L})$ consists of:

– E a finite set of unique event identifiers;
– $\Sigma \subseteq U$ a finite set of activities;
– $\# : E \to (A \not\rightarrow U)$ obtains the attribute values recorded for an event;
– $\mathscr{L} \subseteq E^*$ the set of traces over E. A trace $\sigma \in \mathscr{L}$ records the sequence of events for one process instance. Each event occurs only in a single trace.

Given an event $e \in E$, we write $\#_a(e) \in U$ to obtain the value $u \in U$ recorded for attribute $a \in A$. We require events to record at least the activity attribute: $\#_{act}(e) \in \Sigma$ is the *name of the activity* that caused the event. Given a trace $\langle e_1, \ldots, e_n \rangle \in \mathscr{L}$, we define $val : E \to (A \not\rightarrow U)$ to collect the *latest attribute values* recorded before an event occurred, i.e., $val(e_i) = val(e_{i-1}) \oplus \#(e_{i-1})$ with special case $val(e_1) = f_\varnothing$.[2] We denote

[2] $f \oplus g$ denotes the overriding union of f and g, and $f_\varnothing : \varnothing \to U$ is the empty function.

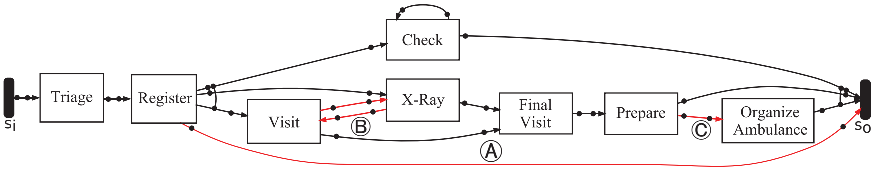


Fig. 3. A causal net (C-net) of the example process. Activities are depicted with boxes, the dependency relations as edges, and the binding functions as black dots on the edges. The unique start and end activities are shown as black boxes. The red edges are explained in Sect. 5.

the *predecessor event* in the trace by ${}^\bullet(e_i) = e_{i-1}$ with special case ${}^\bullet(e_1) = \bot$. Finally, in the remainder of this paper, we assume that a particular event log $L = (E, \Sigma, \#, \mathscr{L})$ exists to avoid unnecessary notation.

Example 1. Table 1 shows three traces $\sigma_1, \sigma_2, \sigma_3 \in \mathscr{L}$ based on the process shown in Fig 1. Each event has a unique identifier. We can identify the activity of event e_{11} as $\#_{act}(e_{11}) = \mathit{Triage}$. Moreover, event e_{11} writes the attribute value $\#_{Priority}(e_{11}) = \text{Red}$. We obtain the latest attribute values recorded before e_{18} occurred as $val(e_{18}) = f$, with $f(\mathit{Priority}) = \text{Red}$ and $f(\mathit{Nurse}) = \text{Joe}$.

Our method uses **Causal nets (C-nets)** to represent the discovered process model [8,22]. A C-net is a tuple $(\Sigma, s_i, s_o, D, I, O)$ where:

- Σ is a finite set of activities;
- $s_i \in \Sigma$ is the unique start activity;
- $s_o \in \Sigma$ is the unique end activity;
- $D \subseteq \Sigma \times \Sigma$ is the dependency relation;
- $B = \{X \subseteq \mathscr{P}(\Sigma) \mid X = \{\varnothing\} \vee \varnothing \notin X\}$ are possible bindings;[3]
- $I \in \Sigma \rightarrow B$ is the set of input bindings per activity;
- $O \in \Sigma \rightarrow B$ is the set of output bindings per activity,

such that the dependency relations match the input and output bindings, i.e., $D = \{(s_1, s_2) \in \Sigma \times \Sigma \mid s_1 \in \bigcup_{\beta \in I(s_2)} \beta \wedge s_2 \in \bigcup_{\beta \in O(s_1)} \beta\}$. We require C-nets to have a unique start and end activity, i.e., $\{s_i\} = \{s \in \Sigma \mid I(s) = \{\varnothing\}\}$ $\{s_o\} = \{s \in \Sigma \mid O(s) = \{\varnothing\}\}$. The input and output binding functions of a C-net define its language. We describe the C-net semantics by example, the full semantics are described in [22].

Example 2. Figure 3 shows how the example from Figure 1 can be modeled as C-net. Activities are depicted with boxes and dependency relations as edges. There are unique start and end activities: s_i and s_o. Output and input bindings are depicted by black dots on the edges in Figure 3. Bindings indicate which combinations of activities can precede or follow a given activity. Connected dots show activities belonging to the same binding. We abbreviate activity names by using the first letter. For example, after activity s_i, activities T and R follow in a sequence, i.e., $O(s_i) = \{\{T\}\}$, $I(T) = \{\{s_i\}\}$ and $O(T) = \{\{R\}\}$, $I(R) = \{\{T\}\}$. Then, there are multiple alternative choices. Three

[3] $\mathscr{P}(\Sigma)$ denotes the powerset of set Σ.

output bindings are defined for R: $O(R) = \{\{s_o\}, \{C, X\}, \{C, V\}\}$. Each set of activities represents a possible choice of following activities (XOR gateway). Either only s_o, or both C and X, or both C and V need to happen. Activities in the same set can be executed in parallel (AND gateway).

4 Data-driven Process Discovery

The DHM builds on the insight that infrequent but data-dependent process behavior is of great interest to process analysts and, thus, should not be disregarded as noise. We extend the ideas of the HM [8] with a measure for *conditional dependency*.

4.1 Data-aware Dependency Measure

To discover data-dependent behavior in the event log, we make use of classification techniques (e.g., decision trees). More specifically, we rely on binary classifiers predicting directly-follows relations based on attribute values recorded in the event log. We denote these classifiers as *dependency conditions*.

Definition 1 (Dependency conditions). *Given universes of attributes A, values U, and activities $\Sigma \subseteq U$, we define the dependency conditions $C \in (\Sigma \times \Sigma) \to ((A \not\rightarrow U) \to \{0, 1\})$. A* dependency condition *$C_{a,b}(x) = (C(a, b))(x)$ is a binary classifier that predicts whether an event of activity a is directly followed by an event of activity b for the attribute values $x \in (A \not\rightarrow U)$, i.e., $C_{a,b}(x) = 1$ when* **b** *is predicted to directly follow* **a** *and $C_{a,b}(x) = 0$ when a* different *activity is predicted.*

In the remainder of the paper, we denote with **1** a special dependency condition function that returns classifiers predicting 1 regardless of the attribute values, i.e., $\forall a, b \in \Sigma, \forall x \in (A \not\rightarrow U) : \mathbf{1}_{a,b}(x) = 1$. Given a dependency condition, we establish the frequency with which activities are observed to directly follow other activities in the event log when the condition holds. We denote this as: *conditional directly follows*.

Definition 2 (Conditional directly follows relation). *Given activities $a, b \in \Sigma$ and dependency conditions C, we write $a >^{C,L} b$ if and only if an execution of activity a with the latest attribute values x is* directly followed *by an execution of activity b under dependency condition $C_{a,b}(x)$. We denote the frequency of a conditional directly follows relation $a >^{C,L} b$ in the event log as:*

$$\begin{aligned} \left|a >^{C,L} b\right| = |\{e \in E \mid &\#_{act}({}^\bullet(e)) = a \wedge {}^\bullet(e) \neq \perp \wedge \#_{act}(e) = b \\ &\wedge C_{a,b}(val(e)) = 1\}|. \end{aligned}$$

Now, we define a data-aware variant of the dependency measure proposed by the HM.

Definition 3 (Conditional dependency measure). *Given activities $a, b \in \Sigma$ and dependency conditions C. We define $a \Rightarrow^{C,L} b : \Sigma \times \Sigma \to [-1, 1]$ as the strength of the causal dependency from a to b under condition $C_{a,b}$ in the event log:*

$$a \Rightarrow^{C,L} b = \begin{cases} \frac{\left|a>^{C,L}b\right| - \left|b>^{C,L}a\right|}{\left|a>^{C,L}b\right| + \left|b>^{C,L}a\right| + 1} & \textit{for } a \neq b, \\ \frac{\left|a>^{C,L}a\right|}{\left|a>^{C,L}a\right| + 1} & \textit{otherwise.} \end{cases}$$

The intuition behind the data-aware variant of these measures is that a relation (a,b) should be included in the dependency relations of the discovered causal net when it is clearly characterized by a certain dependency condition $C_{a,b}$.

Example 3. Consider an event log L with 50 traces like σ_1, 50 traces like σ_2 and 50 traces like σ_3 as shown in Table 1. We determine the conditional dependency measure $X \Rightarrow^{C,L} V$ from activity X-Ray (X) to activity Visit (V). We assume that condition $C_{X,V}(v)$ returns 1 only if attribute *Nurse* values takes on the value Alice. Then, we obtain the number of times X is directly followed by V under condition $C_{X,V}$ as $\left|X >^{C,L} V\right| = 50$, and the number of times V is directly followed by X under conditions C as $\left|V >^{C,L} X\right| = 0$. Therefore, the conditional dependency measure under conditions C is $X \Rightarrow^{C,L} V = \frac{50-0}{50+0+1} \approx 0.98$. This indicates a strong dependency relation from activity X to activity V under condition $C_{X,V}$. By contrast, if we consider the unconditional dependency measure $X \Rightarrow^{\mathbf{1},L} V$, then we obtain $\frac{50-100}{50+100+1} \approx -0.33$. Thus, when disregarding the data perspective, both activities appear to be executed in parallel.

4.2 Discovering Data Conditions

We described the *conditional directly-follows relation* and the *conditional dependency measure*. We use the latter measure to determine which relations should be included in the C-net. Both concepts rely on discovered *dependency conditions*. Here, we describe how to train a classifier that can be used as dependency condition. We build a set of training instances for every combination of activities $(a,b) \in \Sigma \times \Sigma$.

In the remainder, $\mathbb{B}(X)$ denotes the set of all multi-sets over a set X. We use $X = [a^2, b]$ as a short-hand notation to denote the multi-set $X = [a,a,b]$, and $\uplus$ to denote the sum of two multi-sets, i.e., $X \uplus [b,c] = [a^2, b^2, c]$.

Definition 4 (Training Instances). *Given a source activity $a \in \Sigma$, a candidate activity $b \in \Sigma$, and a dependency threshold $\theta_{dep} \in [0,1]$. Let $a^\bullet \subseteq \Sigma$ be the set of activities s that directly follow a in the event log with an unconditional dependency measure above the threshold θ_{dep}, i.e., $a^\bullet = \{s \in \Sigma \mid a \Rightarrow^{\mathbf{1},L} s \geq \theta_{dep}\}$. We collect those events $X_{L,a,b} \subseteq E$ that directly follow an execution of* **a** *in the event log, and refer to activities in* $\mathbf{a}^\bullet$*, or to the candidate activity* **b***, i.e., $X_{L,a,b} = \{e \in E \mid {}^\bullet(e) = a \wedge \#_{act}(e) \in a^\bullet \cup \{b\}\}$. Function $T_{L,\theta_{dep}} \in (\Sigma \times \Sigma) \to \mathbb{B}((A \not\rightarrow U) \times \{1,0\})$ returns the multi-set of training instances:*

$$T_{L,\theta_{dep}}(a,b) = \biguplus_{e \in X_{L,a,b}} [(val(e), cl(e))] \text{ with } cl(e) = \begin{cases} 1, & \text{for } \#_{act}(e) = b \\ 0, & \text{for } \#_{act}(e) \neq b \end{cases}$$

Conceptually, our method is independent of the used classification algorithm. Concretely, we employ decision trees (C4.5) [23] as an efficient method that result in human interpretable conditions. We build the dependency conditions C by assembling a set of training instances $T_{L,\theta_{dep}}(a,b)$ and training a decision tree for each possible relation $(a,b) \in \Sigma \times \Sigma$. Only good dependency conditions with discriminative power are used later on. We use a score $q(C_{a,b}) \in [0,1]$ to determine the quality of a particular condition $C_{a,b}$. There are many possible performance measures for binary classification algorithm that can be used together with our method. None of the measures is universally accepted, the correct choice depends on the concrete application area.

We opted for Cohen's kappa (κ) [24], which indicates whether the prediction was better than a prediction by chance (i.e., for $\kappa > 0$). Kappa favors a good prediction performance on the minority class, which is a desirable property in our setting. Moreover, it has been recommended for nonparametric binary classifiers, such as C4.5, on data with imbalanced class priors [25]. However, we do *not* claim κ to be the best measure and, thus, foresee other measures to be plugged-in depending on the application area.

Example 4. Consider the dependency threshold $\theta_{dep} = 0.9$ and an event log containing 150 traces, where 50 traces record the same values as σ_1, 50 traces the same values as σ_2 and 50 traces the same values as σ_3. We train a classifier for the dependency condition $C_{X,V}$, i.e., the dependency relation from X-Ray (X) to Visit (V) using the training instances $T_{L,\theta_{dep}}(X,V)$. The training instances are $T_{L,\theta_{dep}}(X,V) = [(v_1, \text{Final Visit})^{50}, (v_2, \text{Visit})^{50}]$ with attribute value functions $v_1(P) = \text{Red}$, $v_1(N) = \text{Joe}$ and $v_2(P) = \text{Red}$, $v_2(N) = \text{Alice}$. Please note that there is no instance with the activity Check (C) since the unconditional dependency measure $X \Rightarrow^{\mathbf{1},L} C$ is below the threshold of 0.9. Therefore, the instances based on trace σ_3 are not included as we already know that activity C is in parallel to X. We train a C4.5 decision tree and obtain the dependency condition $C_{X,V}$ with $C_{X,V}(v_2) = 1$ and $C_{X,V}(v_1) = 0$.

4.3 Data-driven Discovery of Causal Nets

We describe the DHM method that builds C-nets based on *conditional dependencies*. The DHM supports four user-specified thresholds that can be used to tune the noise filtering capabilities to specific needs of the user. All thresholds range between 0 and 1:

- θ_{obs}, the observation threshold, which controls the relative frequency of relations;
- θ_{dep}, the dependency threshold, which controls the strength of causal dependencies;
- θ_{bin}, the binding threshold, which controls the number of bindings;
- θ_{con}, the condition threshold, which controls the quality of data-dependencies.

We discover a C-net $(\Sigma, s_i, s_o, D, I, O)$ from event log $L = (E, \Sigma, \#, \mathscr{L})$ and thresholds $\theta_{obs}, \theta_{dep}, \theta_{bin}, \theta_{con}$ in the following steps.

1. We want to ensure that the resulting C-net has a **unique start and end activity**. Therefore, we add artificial start and end events to all traces, i.e., $\forall_{\sigma \in \mathscr{L}}(\sigma = \langle e_i, e_1, \ldots, e_n, e_o \rangle \wedge \#_{act}(e_i) = s_i \wedge \#_{act}(e_o) = s_o)$ and $\Sigma = \Sigma \cup \{s_i, s_o\}$.
2. We build the set of standard **dependency relations** as follows:
$$D = \{(a,b) \in \Sigma \times \Sigma \mid a \Rightarrow^{\mathbf{1},L} b \geq \theta_{dep} \wedge \frac{|a >^{\mathbf{1},L} b|}{|\mathscr{L}|} \geq \theta_{obs}\}.$$
3. We **discover the dependency conditions** C by training classifiers for each pair $(a,b) \in \Sigma \times \Sigma$ using the training instances $T_{L,\theta_{dep}}(a,b)$.
4. We add the **conditional dependency relations** to D. We use θ_{con} instead of θ_{obs} to obtain infrequent, high-quality data conditions:
$$D = D \cup \{(a,b) \in \Sigma \times \Sigma \mid q(C_{a,b}) \geq \theta_{con} \wedge a \Rightarrow^{C,L} b \geq \theta_{dep}\}.$$

5. Some activities $s \in \Sigma$ might not have a predecessor or successor in the directed graph induced by D. Intuitively, each task in a process should have a cause (predecessor) and an effect (successor) [8], all tasks in **the C-net should be connected**. Therefore, we propose two alternative heuristics to enforce this:
   - *all-task-connected heuristic* proposed by the HM [8], or
   - the *accepted-task-connected heuristic*, a new heuristic.

   Here, we describe the new *accepted-task-connected heuristic*. We repeatedly connect only those activities that are already part of the dependency graph using their best neighboring activities until all activities have a cause and an effect. Then, set $\overline{D}$ of relations necessary to connect all activities accepted so far is:

$$\begin{aligned}\overline{D} = \{(a,b) \in \Sigma \times \Sigma \mid &(\nexists x\ (a,x) \in D \wedge \forall y\ (a \Rightarrow^{\mathbf{1},L} b) \geq (a \Rightarrow^{\mathbf{1},L} y)) \\ &\vee (\nexists x\ (x,b) \in D \wedge \forall y\ (a \Rightarrow^{\mathbf{1},L} b) \geq (y \Rightarrow^{\mathbf{1},L} b))\}.\end{aligned}$$

   We extend the dependency relations with the new relations, i.e., $D = D \cup \overline{D}$. There might be new, unconnected activities in D. Therefore, we *repeat adding the best neighboring activities* until set $\overline{D}$ is empty.
6. We discover the **input and output binding functions** of the C-net. For the output binding function $O(a)$ of an activity $a \in \Sigma$, we need to determine which executions of $b \in \Sigma$ (with $(a,b) \in D$) were caused by an execution of activity **a**. We use the heuristic proposed by the HM [8] and repeat it for completeness. The heuristic considers activity **b** to be caused by activity **a** only if it is the nearest activity that *may have caused* **b**. Any other activity **s** executed in between **a** and **b** should not be a possible cause of **b**, i.e., $(s,b) \notin D$. Given a trace $\sigma = \langle e_1, \ldots, e_i, \ldots, e_n \rangle \in \mathscr{L}$, the set of activities $\overline{O}(e_i) \subseteq \Sigma$ that were caused by an event e_i is:

$$\begin{aligned}\overline{O}(e_i) = \{b \in \Sigma \mid &\#_{act}(e_i) = a \\ &\wedge \exists_{i<j\leq n}\ \#_{act}(e_j) = b \wedge (a,b) \in D \\ &\wedge \forall_{i<k<j}\ (\#_{act}(e_k), b) \notin D\}.\end{aligned}$$

   We determine the frequency $|o|_{L,a} \in \mathbb{N}$ of an output binding $o \subseteq \Sigma$ for activity $a \in \Sigma$ in the event log L as:

$$|o|_{L,a} = \left|\{e \in E \mid \#_{act}(e) = a \wedge \overline{O}(e) = o\}\right|.$$

   Then, we build the complete multi-set of output bindings with the most frequent bindings. Those bindings that fulfill the user-specified binding threshold θ_{bin}:

$$O(a) = \{o \subseteq \Sigma \mid \frac{|o|_{L,a}}{max_{\overline{o} \subseteq \Sigma}(|\overline{o}|_{L,a})} \geq \theta_{bin}\}.$$

   The input binding function I is obtained by reversing the same approach.

Within the scope of this paper, we do not elaborate on the other heuristics of the HM [8], such as long-distance, length-two loops, and the relative-to-best. Those heuristics and improvements to the HM described by the Fodina miner [26] can be used together with the DHM. The choice which heuristics to apply highly depends on the process at hand. For example, the *all-task-connected heuristic* results in a process model with all observed activities regardless of the chosen observation frequency threshold θ_{obs}. Even activities that are only observed once are added. This might not be desirable as very infrequent activities might be considered as noise. Therefore, we introduced the new accepted-task-connected heuristic.

5 Evaluation

We implemented the DHM in the open-source framework ProM[4]. The package *DataAwareCNetMiner* provides a *highly interactive tool*, which allows to quickly discover C-nets for different parameter settings and to explore the discovered data dependencies. C-nets can be converted to Petri nets or BPMN models. Therefore, existing tools can be used on the results. We applied our method to both synthetic and real-life event logs.

5.1 Synthetic - Handling Noise

Event Log & Methods. We generated an event log with 100,000 traces and approximately 900,000 events by simulating the process model shown in Fig 3. There are three data attributes: Priority (P), Nurse (N), and Type (T). We adjust the frequency distributions of these attributes such that paths A, B, and C in model Figure 3 are recorded infrequently. Specifically, only 1.4% of the traces record $P = white$, 19.1% of the traces record $N = Alice$, and 4.3% of the traces record $T = out$. We compared three methods: our proposed method (DHM), the heuristic miner with frequency filtering (HMF), and the heuristic miner without frequency filtering (HMA). All three methods, used thresholds $\theta_{obs} = 0.06$ (0.0 for HMA), $\theta_{dep} = 0.9$, $\theta_{bin} = 0.1$, $\theta_{con} = 0.5$ together with the *accepted-task-connected heuristic*. We used C4.5 as classifier and estimated its performance using 10 times 10-fold cross validation.

Experimental Design. The experiment should evaluate the noise filtering capabilities of our method. Therefore, we injected noise into the event log by randomly adding *one additional event* to an increasing number of traces.[5] Then, we compared the discovered dependency relations with those of the reference model (Figure 3) in terms of graph edit distance (GED) [27]. We did not use fitness, precision, or behavioral comparison measures as those would not be applicable in this setting. Fitness and precision do not measure the performance wrt. the reference model (gold standard). Moreover, when the discovered models are not sound (e.g., having a deadlock), the behavior may be undefined even when the model is close to the original. Behavioral measures would also fail to distinguish the difference between the data-dependent re-sequencing of activities (pattern Ⓒ in Figure 3) and simple parallelism. For example, both in Figure. 3 and in Figure 2(b) activities `Visit` and `X-Ray` are behaviorally in parallel.

[4] The package *DataAwareCNetMiner* can be downloaded from `http://promtools.org`.

[5] The synthetic event logs can be downloaded from `http://dx.doi.org/10.4121/uuid:32cad43f-8bb9-46af-8333-48aae2bea037`.

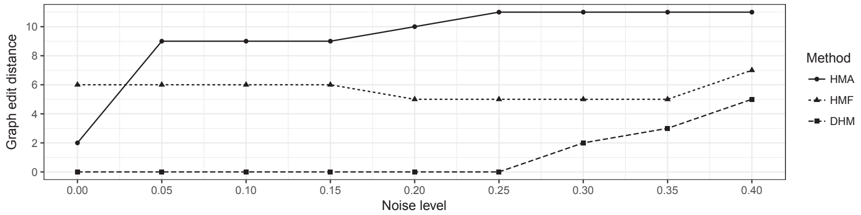


Fig. 4. Graph edit distances between the dependency relations discovered by the compared methods and the reference dependency relations for varying amounts of injected noise.

Results. All models could be discovered in about 3 seconds using 2 GB of memory. Our method was able to rediscover the conditional relations Ⓐ, Ⓑ and Ⓒ, i.e., the red edges in Figure 3. The original rules $P = \mathit{white}$ and $T = \mathit{out}$ were discovered for relations Ⓐ and Ⓑ. For path Ⓒ, two rules were discovered: $N = \mathit{Alice}$ for the edge from X-Ray to Visit and $N \neq \mathit{Alice}$ for the edge from Visit to X-Ray. Our method discovered the data-dependent re-sequencing of activities Visit to X-Ray, whereas the standard HM (cf. BPMN model in Figure 2(b)) considered both activities as parallel. Figure 4 shows the result of the GED measurement for noise levels ranging from 0% to 40%. Our method (DHM) handles the added noise well until 25% of the traces were modified. The HM with frequency-based noise filtering (method HMF) is also unaffected by the injected noise. However, it fails to discover the reference model even without noise, as shown in Figure 2(b). The GED of the method HMF improves after injecting noise in 20% of the traces because the frequency of relation Ⓒ increases by chance. When lowering the observation frequency threshold (method HMA), the injected noise quickly affects the discovery and undesirable dependencies appear. We did not include the IM in Figure 4, as it returns models with a different structure. However, the models returned by the IM are already undesirable for an event log with 5% noise, c.f., Figure 2(a).

5.2 Real-life - Revealing Data dependencies

We used two real-life event logs to show that our method can reveal infrequent behavior in a practical setting. Using the DHM important conditional dependencies can be found where existing methods abstract away such dependencies.

Data and methods. The *Road Fines* (RF) event log was recorded by an information system that handles road-traffic fines by an Italian local police force [28,29]. This event log contains about 150,000 cases, 500,000 events, and 9 data attributes. The *Hospital Billing* (HB) event log was obtained from the ERP system of a hospital. It contains 100,000 cases with 550,000 events and 38 data attributes related to the billing of medical services. We applied the proposed method (DHM), the HM with the same frequency filter settings (HMF) and the Inductive Miner (IM) to both event logs. Without a reference model and knowledge about expected noise levels, we could not compare the discovered models to a gold standard. Therefore, we compare the novel insights obtained by using our method with those from the other methods.

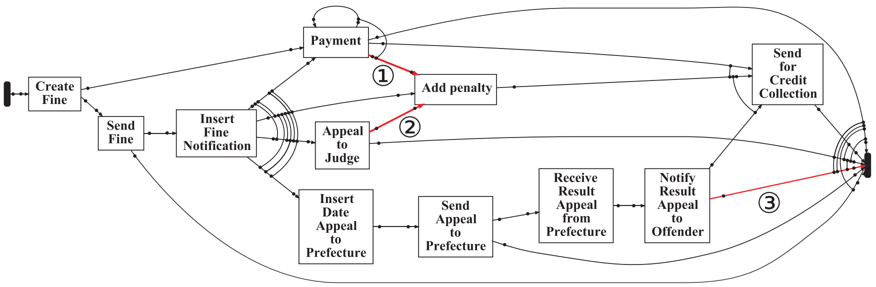

Payment
Create Fine
Send Fine
Insert Fine Notification
Add penalty
Appeal to Judge
①
②
③
Send for Credit Collection
Insert Date Appeal to Prefecture
Send Appeal to Prefecture
Receive Result Appeal from Prefecture
Notify Result Appeal to Offender

Fig. 5. Process model discovered for the RF log. The numbered edges were added by our method.

Table 2. Dependency conditions discovered for the RF log.

Nr	Source	Target	Count	Quality	Dependency	Used Attributes
1	Appeal to Judge	Add penalty	279	0.86	0.93	amount, dismissal, points, article
2	Payment	Add penalty	3,629	0.89	1	amount, isPaid
3	Not. Res. Appeal to Off.	End	83	0.56	0.98	dismissal, expense

Road Fines. Figure 5 shows the C-net discovered by the DHM in about 4 seconds for the RF log. We used the *all-task-connected heuristic* of the original HM, since we know that each activity is of interest. We used eight of the attributes including a derived `isPaid` attribute since this process is about the payment of fines. We used C4.5 with 10-fold cross validation and only accepted classifications with $\theta_{con} \geq 0.5$. Most of the observed behavior (97.8%) can be replayed on the C-net using the alignment method presented in [22]. Our method reveals three additional relations (red edges), which are numbered in Figure 5. Table 2 lists the conditional data-dependency measure, the frequency, as well as quality and used attributes of the obtained dependency condition for each relation. The first two relations target activity *Add Penalty* and both have a very good quality score. The decision rule for relation ① mainly depends on the value of the `dismissal` attribute. Cases with values `G` do not receive a penalty, whereas cases with value `NIL` receive a penalty depending on the fine amount, the number of points, and the article. According to [29] this is to be expected as those cases are dismissed by the judge. Relation ② is mainly based on the attribute `isPaid`. Unpaid fines that have with a small `amount` of less than 35 EUR receive a penalty. Relation ③ was discovered for cases with a `dismissal` value of # or G. It is to be expected that the process finishes for cases with this code, since those cases are dismissed by the prefecture. Interestingly, this relation also occurs for cases with a `dismissal` value of NIL and high postal expenses. This should not happen, since those fines still need to be paid [29]. The DHM revealed three data dependencies that give more insights into the recorded behavior without obstructing the process model with infrequent noise. In the model obtained by IM none of the three relations are directly visible. Therefore, current decision mining techniques would not be able to discover the conditions.

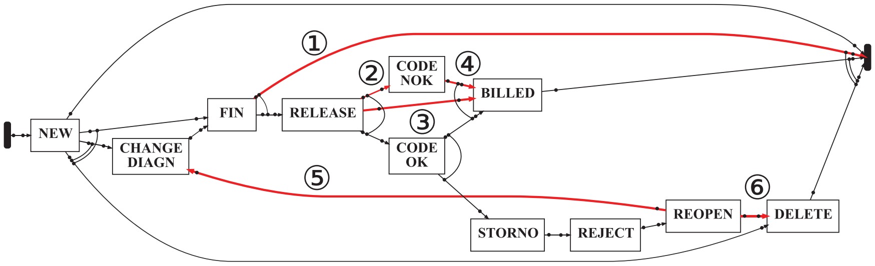


Fig. 6. Process model discovered for the HB. The numbered edges are added by our method.

Table 3. Dependency conditions discovered for the HB log.

Nr	Source	Target	Count	Quality	Dependency	Used Attributes
1	Fin	End	3,619	0.98	1	closeCode
2	Release	Code Nok	1,674	0.62	0.99	caseType
3	Release	Billed	468	0.93	0.98	caseType
4	Code Nok	Billed	1,481	0.84	0.99	caseType, specialty
5	Reopen	Delete	1,128	0.83	0.81	closed
6	Reopen	Change Diagn	212	0.97	0.99	closed

Hospital Billing. Figure 6 shows the C-net discovered by the DHM in about 3 seconds for the HB event log. The discovered model fits 97% of the observed behavior. We used the new *accepted-task-connected heuristic* since not all of the 21 activities may be of of interest. We discovered the model using C4.5 on a subset of 13 attributes. Here, the quality threshold is set to $\theta_{con} \geq 0.6$ and the quality is, again, determined by 10-fold cross validation. Compared to the model returned by the HMF, our method revealed six additional dependencies. Again, we numbered these relations in Figure 6, and list some key statistics in Table 3. For the purpose of this evaluation, we discussed the discovered conditional dependencies with a domain expert from the hospital who works in this process. Relation ① is based on a special `closeCode` that is used when nothing can be billed and, hence, the process ends. Relation ② occurs mostly for two specific `caseType` values. According to the domain expert both case types correspond to exceptional cases: one is used for intensive care and the other for cases for which codes cannot be obtained (*Code Nok*). Relation ③ is, again, related to a specific `caseType`. This type is used for intensive-care activities as well and, often, does not require a code to be obtained. Relation ④ is also mainly related to the `caseType` and to some degree to the medical `specialty`. Both relation ⑤ and relation ⑥ are conditional to the attribute `closed`, which indicates whether the invoice is closed or not. Clearly, deleted cases should not be in the closed status, whereas reopened cases with a change in diagnosis can be eventually closed in the future. The process model discovered by the DHM provides a balanced view on the interesting infrequent paths of the billing process together with the more frequent, regular behavior. Moreover, additional insight is provided by revealing the conditions with which infrequent paths occur. Again, the model returned by the IM did not include any of the six paths.

Limitations We acknowledge that there are some limitations to our method. First, we only consider conditional directly-follows dependencies. Like most process mining approaches, our method requires sufficiently large event logs. Small event logs might, by chance, not contain all directly-follows relations. Moreover, more complex patterns of conditional infrequent behavior, e.g., longer sequences or sub-processes, cannot be discovered. Second, there is a risk that the returned C-nets are unsound [22] since our method is based on the HM. However, recent research shows that it is possible to structure the discovered model afterwards [30]. Third, as all data-driven method the DHM relies on data attributes and infrequent process paths being recorded. Last, we used only two real-life event logs in the evaluation. Therefore, only limited claims on the general applicability of the method can be made. We have also tested the DHM on other event logs. However, very few event logs with data attribute are publicly available.

6 Conclusion

We presented the Data-aware Heuristic Miner (DHM), a *process discovery* method that *reveals conditional infrequent behavior* from event logs. The DHM distinguishes undesired noise from infrequent behavior that can be characterized by conditions over the data attributes of the event log. This is the *first approach that uses both event labels and data attributes when discovering the control-flow*. Dependency conditions are discovered using classification techniques, and, then, embedded in a complete process discovery algorithm built upon the Heuristic Miner. The returned process models are annotated with information on the discovered rules. We applied the DHM to a synthetic and two real-life events logs of considerable size and complexity. We showed that the DHM can efficiently handle large event logs and is robust against typical levels of random noise. The evaluation on two real-life cases shows that the DHM provides insights that could be easily missed when relying on state-of-the-art, frequency-based techniques. In our future work, we would like to extend the idea from directly-follows relations to more complex patterns of conditional behavior (e.g., long-term dependencies). The DHM successfully reveals data dependencies based on directly-follows relations, but dependencies that cannot be captured by directly-follows relations might be missed.

References

1. Davies, I., Green, P., Rosemann, M., Indulska, M., Gallo, S.: How do practitioners use conceptual modeling in practice? Data Knowl. Eng. **58**(3) (2006) 358–380
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
3. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Inf. Syst. **37**(7) (2012) 654–676
4. Suriadi, S., Andrews, R., ter Hofstede, A., Wynn, M.: Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. Information Systems **64** (2017) 132 – 150
5. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowledge Data Eng. **16**(9) (2004) 1128–1142

6. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In: BPM. Volume 5240 of LNCS., Springer (2008) 358–373
7. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In: BPM. Volume 4714 of LNCS., Springer (2007) 328–343
8. Weijters, A., Ribeiro, J.: Flexible heuristics miner (FHM). In: CIDM, IEEE (2011) 310–317
9. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM Workshops. Volume 171 of LNBIP., Springer (2013) 66–78
10. Liesaputra, V., Yongchareon, S., Chaisiri, S.: Efficient process model discovery using maximal pattern mining. In: BPM. Volume 9253 of LNCS., Springer (2015) 441–456
11. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. J. Mach. Learn. Res. **10** (2009) 1305–1340
12. Ponce de León, H., Carmona, J., vanden Broucke, S.K.L.M.: Incorporating negative information in process discovery. In: BPM. Volume 9253 of LNCS., Springer (2015) 126–143
13. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: IEEE Congress on Evolutionary Computation, IEEE (2012) 1–8
14. Rembert, A.J., Omokpo, A., Mazzoleni, P., Goodwin, R.: Process discovery using prior knowledge. In: ICSOC. Volume 8274 of LNCS., Springer (2013) 328–342
15. Bellodi, E., Riguzzi, F., Lamma, E.: Statistical relational learning for workflow mining. Intell. Data Anal. **20**(3) (2016) 515–541
16. Ghionna, L., Greco, G., Guzzo, A., Pontieri, L.: Outlier detection techniques for process mining applications. In: ISMIS. Volume 4994 of LNCS., Springer (2008) 150–159
17. Conforti, R., Rosa, M.L., t. Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. **29**(2) (Feb 2017) 300–314
18. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering simulation models. Inf. Syst. **34**(3) (2009) 305–327
19. de Leoni, M., van der Aalst, W.M.P.: Data-aware process mining: Discovering decisions in processes using alignments. In: SAC'13, ACM (2013) 1454–1461
20. Bazhenova, E., Bülow, S., Weske, M.: Discovering decision models from event logs. In: BIS. Volume 255 of LNBIP., Springer (2016) 237–251
21. Schönig, S., Di Ciccio, C., Maggi, F.M., Mendling, J.: Discovery of multi-perspective declarative process models. In: ICSOC. Volume 9936 of LNCS., Springer (2016) 87–103
22. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Causal nets: A modeling language tailored towards process discovery. In: CONCUR. Volume 6901 of LNCS., Springer (2011) 28–42
23. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
24. Cohen, J.: A coefficient of agreement for nominal scales. Educ. Psychol. Meas. **20**(1) (1960) 37–46
25. Ben-David, A.: About the relationship between ROC curves and cohen's kappa. Eng. Appl. Artif. Intell. **21**(6) (2008) 874 – 882
26. vanden Broucke, S.: Advances in Process Mining: Artificial negative events and othertechniques. PhD thesis, KU Leuven (2014)
27. Dijkman, R.M., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In: BPM. Volume 5701 of LNCS., Springer (2009) 48–63
28. de Leoni, M., Mannhardt, F.: Road traffic fine management process (2015) doi:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.
29. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing **98**(4) (2016) 407–437
30. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Bruno, G.: Automated discovery of structured process models: Discover structured vs. discover and structure. In: ER. Volume 9974 of LNCS. (2016) 313–329