Causal Reasoning over Control-Flow Decisions in Process Models

Sander J.J. Leemans[1][0000−0002−5201−7125] and Niek Tax[2][0000−0001−7239−5206]

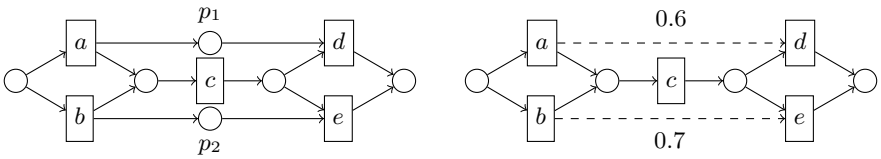
[1] Queensland University of Technology, Brisbane, Australia
[2] Meta, London, United Kingdom
s.leemans@qut.edu.au, niek@fb.com

Abstract Process mining aims to provide analysts with insights, such that business processes supported by information systems can be improved. Traditionally, insights from process mining projects and techniques have been associational rather than causal, thus only describing the current state of the process, without predictive capabilities over effects of hypothetical process changes, which inherently limits business process optimisation efforts. In this paper, we introduce causal analysis for control-flow decisions taken during the execution of process models: using an event log and the structure of a process model, we (i) extract the set of decision points in the process, (ii) apply a causal discovery approach to obtain a collection of causal graphs that are consistent with the observations in the event log, (iii) extract ordered pairs of decision points between which a causal connection can be ruled out based on the temporal ordering that is implied by the process model specification, and use these to narrow down the set of possible causal graphs. This technique addresses the problem of mining dependencies, which has long been a challenge in the process discovery field. The technique has been implemented in the Visual Miner as part of the ProM framework. We illustrate the technique using examples and demonstrate its applicability on real-life logs.

Keywords: Process mining · causal discovery · intra-model dependencies · long-distance dependencies

1 Introduction

Process mining aims to gain insights into business processes of organisations from event data recorded in information systems. Typical process mining projects aim to gain such insights such that the process can subsequently be improved, for instance to reduce cost, reduce cycle time or increase efficiency. Gaining insights into business processes can be supported by several automated process mining techniques, such as the discovery of a process model from recorded event data, the study of conformance checking and the projection of data such as costs, performance information and other data on process models. Using such enriched process models, process mining techniques have been developed that recommend interventions or optimal pathways through the process for ongoing cases by predicting process outcomes.



(a) Deterministic dependencies. (b) Probabilistic dependencies.

Figure 1: A Petri net with dependencies.

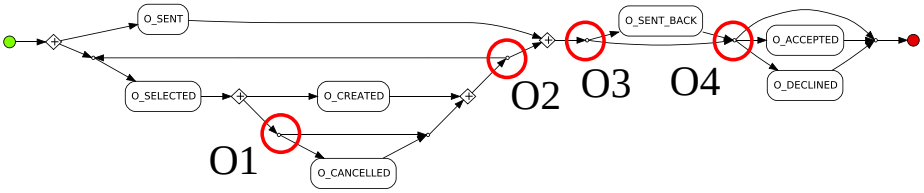
Typical process mining techniques are inherently *associational*, that is, they merely describe or visualise the data and cannot be used to reason in retrospect (i.e. counterfactual) or to identify *causes* of identified phenomena. For instance, a negative association between identified fraud cases and the performed thorough checks earlier in a process does not suffice to conclude that the number of identified fraud cases can be lowered by performing more thorough checks earlier in the process. Similarly, a positive association between the sales of ice cream and deaths by drowning does not suffice to conclude that deaths by drowning can be prevented by lowering the sales of ice cream. Thus, such associational insights have limited value in assisting redesigns of the process.

Causal reasoning has been leveraged to, for instance, predict the influence of trace-based interventions on cycle time [4], to influence the likelihood of a given outcome [3], and to explain why a certain negative outcome was achieved in a particular trace [22]. However, none of these causal process mining techniques are tailored towards process discovery, make the connection to (long-distance) dependencies, or in any way leverage process models.

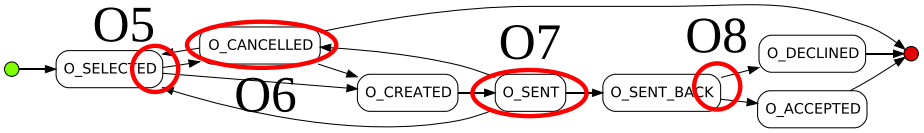
An often overlooked aspect of process models is *dependencies*: the dependence of decisions in the process on earlier decisions. For instance, the Petri net in fig. 1a contains two long-distance dependencies: d can only be executed if a was executed earlier in the process and e can only be executed if b was executed earlier in the process. As d cannot be executed without a having been executed, this is a *deterministic* dependency, enforced by places p_1 and p_2. Rather, in this paper, we consider *probabilistic* dependencies. For instance, the Petri net in fig. 1b is annotated with long-distance dependencies indicating that after execution of a, the probability of executing d is 60%, while after execution of b, this probability is 30%. Obviously, Petri nets can only capture deterministic dependencies. For process discovery, we argue that we might still want to visualise probabilistic dependencies to the process analyst *as long as these dependencies are causal*.

In this paper, we aim to identify causal relationships among decision points in the process. The motivation behind these causal relationships includes the following use cases:

Informing process interventions and design. If activity d in fig. 1b affects the business positively (e.g., a purchase of a certain product) while e represents a negative outcome (e.g., ending the process without a purchase), then it is useful to know whether the decision between a and b has a causal effect on the decision between d and e. If this relation is causal, then information about this causal



(a) Visualisation of a process tree.



(b) DFM.

Figure 2: Two models of the BPI Challenge 2012 log - O_ activities. Red circles indicate option sets.

relation could influence process participants' decisions as well as process redesign efforts, e.g. by preferring a over b.

Discovering long-distance dependencies in process models. The control flow perspective of process models is inherently causal: the execution of a transition may enable other transitions and is thus, according to the model, causal. For instance, in fig. 1a, executing a has a causal influence on whether d is executed later on in the process. It is thus natural to extend the model with only causal probabilistic dependencies.

We introduce a method to study causal control-flow dependencies in process models. Using this causal knowledge, control-flow decisions in processes that could be leveraged to influence the decision can be identified. To this end, given a process model and an event log, our technique derives the causal structure of decision points in the model, after which causal reasoning is applied to provide analysts with insights into the causal dependencies in a process. The method has been implemented and we evaluate it using synthetic and real-life examples, and we demonstrate its applicability on real-life logs.

This paper is organised as follows: section 2 introduces a motivating example; section 3 discusses related work; section 4 introduces existing concepts; section 5 introduces our method; section 6 evaluates it and section 7 concludes the paper.

2 Motivating Example

As an example, we study the choices made in a loan application process (BPI Challenge 2012 - `O_` activities). To this log, we applied a discovery technique to obtain a process tree (see fig. 2a). This model has four decisions points (*option sets*), labelled o1-o4. Option sets o1 and o2 form a loop: the options of o2 entail either exiting the loop or doing it again, and o1 is part of that loop. For every time a trace traverses the loop, a decision must be made on o1 and o2. Thus, each trace might have multiple *choices* corresponding to the option sets o1 and o2. That is, the option sets that are part of the loop can be *unfolded* into *choices*. We formalise option sets and choices in Section 5.1.

Table 1: Some probabilistic dependencies of the model in fig. 2a.

(a) O1 vs. O1 (unfolding 2) $\chi^2 = 642$, $p < 0.00001$

	o_cancelled	skip
o_cancelled	278	462
skip	698	0

(b) O1 vs. O2 $\chi^2 = 129$, $p < 0.00001$

	[loop redo]	[loop exit]
o_cancelled	740	1222
skip	698	2355

(c) O1 vs. O2 (unfolding 2) $\chi^2 = 0.58$, $p = 0.45$

	[loop redo]	[loop exit]
o_cancelled	205	535
skip	206	492

To study probabilistic dependencies, Table 1 shows how often certain decisions appeared together. If in the first unfolding of O1, `o_cancelled` is chosen, then in O3, `o_sent_back` is chosen 406 times (21%), while `o_sent_back` is skipped 1556 times (79%). However, if `o_cancelled` is skipped in the first unfolding of O1, then in O3, `o_sent_back` is executed 2568 times (84%) and skipped 485 times (16%). With a Benjamini-Hochberg correction [2] for the 14 tests performed (assuming at most 2 unfoldings), a χ^2 test confirms that this difference is statistically significant. Yet, this is insufficient information for stakeholders to decide how to best decrease the relative number of loan applications that are sent back (O3), since this is not yet shown to be a *causal* relation. In the remainder of this paper, we study these two models as running examples.

3 Related Work

Long-Distance Dependencies. Many discovery techniques explicitly support long-distance dependencies, including the Flexible Heuristics Miner [32], Fodina [5], some variants of the α algorithm [33], the ILP algorithm [34] and Declare [19]. Most of these techniques apply a threshold to decide on the presence of a long-distance dependency and thus still produce deterministic dependencies, or only include long-distance dependencies if they do not violate fitness at all (e.g. ILP), thus all produce only deterministic dependencies. An exception is [27], which considers probabilistic dependencies through association rules in one particular Petri net structure (though not causal). Thus, while deterministic dependencies in models have been studied (e.g. ILP, HM, Fodina), to the best of our knowledge, causal dependencies have not.

Some discovery techniques are limited by their representational bias in representing long-distance dependencies. For instance, process trees cannot represent long-distance dependencies due to their block structure [13]. As the state of directly follows-based models consists only of the last executed state, these models cannot represent long-distance dependencies either [9,16]. BPMN models can represent a limited set of long-distance dependencies, due to a lack of an explicit state and a waiting-less semantics.

Prediction. Given an enriched process model, several approaches have been proposed that aim to predict certain aspects of ongoing cases, such as the likelihood of undesirable events [6], timeliness [29,12,10], the next executed activity [30], the utilisation of resources [20] or in general event data [7]. While a recent approach [11] uses counterfactual reasoning to explain predictions, these approaches to prediction do not provide information on intervention (what-if), for which a causal approach is necessary,

Causal Analysis in Process Mining. Causal inference has been used to determine the influence of process steps on the outcomes of ongoing traces [24], and

to maximise the likelihood of a desirable outcome for a running case [3]. Similar approaches have been proposed to decrease service time [21] or cycle time [4] of a case, or the root causes of performance issues [26]. Counterfactual reasoning has been applied to process mining as well, to explain why an undesirable outcome was achieved for a particular case [22]; a diversified range of potential scenarios is presented to the analyst.

Bayesian network structure learning from event logs is proposed in [28,29]. Bayesian networks can in principle be causal models if their structure coincides with the structure of the causal graph, however, the Bayesian networks in [28,29] are not causal, as they focus on optimising the structure for predictive accuracy rather than on attempting to find causal relations.

To the best of our knowledge, only two causal approaches in process mining advocate the use of a process model: [17] uses Structural Causal Models to answer what-if questions by quantifying the improvement of proposed process changes; [17] mentions a broad range of causal principles, including using the structure of process models. Both techniques however provide no guidance or detail on how to apply these, and do not consider the causal dependencies between model choices. Similarly, [23] proposes root-cause analysis and mentions the potential use of process models, but does not detail how process models can be leveraged.

More generally, while existing work in the area of causality in process mining focuses on identifying or quantifying causal effects on process outcomes or process performance, this work, in contrast, focuses on causal effects of control-flow decisions on other control-flow decisions later in the process. Thereby, this work is positioned at the intersection of causal inference and process discovery, rather than at the intersection of causal inference and business process improvement.

4 Preliminaries

In this section, we introduce existing concepts and notations.

A *multiset* is a function from a set of elements Σ to the natural numbers, indicating how often each element appears in the multiset. For instance, $[a^2, b^7]$. For a multiset M, $[. \mid .]$ denotes multiset composition, such that $[a \mid a \in M] = M$.

A *trace* is a sequence of events, representing the activities executed for a particular case in a process. An *event log* is a multiset of traces. For instance, $[\langle a, b, c\rangle^2, \langle a, d\rangle^3]$ represents a log with 5 traces, of which 2 traces have 3 activities. The empty trace is denoted with ϵ; the set of all logs is $\mathcal{L}$.

$\mathcal{M}$ is the set of all process models (regardless of formalism). A *directly follows model* (DFM) is a directed graph consisting of transitions T and edges E, such that $start \notin T$ and $end \notin T$. A DFM expresses a set of traces as each trace starts in *start* and moves over the edges to *end*, executing the activities annotated on the transitions along the path. See [16] for a full formal definition. Figure 2b shows an example of a DFM, which supports the trace $\langle$`o_selected`, `o_cancelled`$\rangle$ amongst other traces.

A *process tree* is a block-structured process model, defined recursively [13]. Each node in the tree describes a language; a *leaf* $a \in \Sigma$ describes the singleton language of its activity, a *silent leaf* τ) describes the language with the empty trace $\{\epsilon\}$ and a *node* describes a combination of the behaviour of its sub-trees

using an operator $\oplus$. In this paper, we consider six n-ary operators: the sequential composition $\mapsto$, exclusive choice $\times$, inclusive choice $\vee$, interleaved $\leftrightarrow$, concurrent $\wedge$, and $\circlearrowleft$; where $\circlearrowleft(T_1, T_2, T_3)$ combines three sub-trees as an always-executed body T_1, and then a choice between executing T_2 followed by T_1 and back to the same choice, or exiting the loop by executing T_3. For a formal definition, please refer to [13]. As an example, fig. 2a shows a visualisation of the process tree $\mapsto(\wedge(\text{o\_sent}, \circlearrowleft(\mapsto(\text{o\_selected}, \wedge(\text{o\_created}, \times(\tau, \text{o\_cancelled}))), \tau, \tau)), \times(\tau, \text{o\_sent\_back}), \times(\tau, \text{o\_accepted}, \text{o\_declined}))$.

A *structural causal model* [18] over a set of variables V is a system of equations of the form $v_i = f_i(\text{pa}(v_i), U_i)$ for all $v_i \in V$, where $\text{pa}(v_i)$ denotes the set of variables that directly determine the value of v_i (i.e., the parents), and U_i represents errors that might arise as a result of either true randomness (i.e., a coin flip) or residual errors due to omitted variables. A *causal graph* [18] of a set of variables V is a directed acyclic graph consisting of nodes V and edges $E = \{(v_i, v_j) \in V \times V \mid v_i \in \text{pa}(v_j)\}$, i.e., the causal graph denotes the *structural form* of the structural causal model without specifying the *functional form*. We write $v_i \to v_j$ for $(v_i, v_j) \in E$, representing that v_i has a causal effect on v_j. Furthermore, we write $v_i \text{ — } v_j$ for $v_i \to v_j \vee v_i \leftarrow v_j$.

Random variables X and Y are *conditionally independent* given a set of random variables Z, denoted $X \perp\!\!\!\perp Y \mid Z$, if and only if $P(X \mid Y, Z) = P(X \mid Z)$, i.e., given that we already know Z, knowing additionally Y provides no addition information about X, and vice versa[3]. When $Z = \emptyset$, i.e., $P(X \mid Y) = P(X)$, X and Y are *marginally independent*, denoted $X \perp\!\!\!\perp Y$.

A causal graph G on variables V and a probability distribution $P(v_1, \ldots, v_{|V|})$ satisfies the *causal Markov condition* if and only if $\forall_{v_i, v_j \in V, v_i \neq v_j} \Rightarrow v_i \perp\!\!\!\perp v_j | \text{pa}(v_i)$, i.e. each variable is independent of all its non-descendants given its parents G.

A *path* on a causal graph $G = (V, E)$ is a sequence of distinct vertices $\langle v_1, \ldots v_n \rangle$ such that $\forall_{1 \leq i < n} v_i \text{ — } v_{i+1}$. A *directed path* is a path such that $\forall_{1 \leq i < n} v_i \to v_{i+1}$. A vertex v_i on a path $\langle v_1, \ldots v_n \rangle$ is called (i) a *chain* if $v_{i-1} \to v_i \to v_{i+1}$ or $v_{i-1} \leftarrow v_i \leftarrow v_{i+1}$; (ii) a *collider* if $v_{i-1} \to v_i \leftarrow v_{i+1}$ (i.e., v_i is a common effect of its neighbours); and (iii) a *confounder* if $v_{i-1} \leftarrow v_i \to v_{i+1}$ (i.e., v_i is a common cause of its neighbours).

A path p in causal graph G is *d-separated* [8,18] by $Z \subseteq V$ if and only if either p contains a chain $v_{i-1} \to v_i \to v_{i+1}$ or confounder $v_{i-1} \leftarrow v_i \to v_{i+1}$ such that $v_i \in Z$, or p contains a collider $v_{i-1} \to v_i \leftarrow v_{i+1}$ such that $v_i \notin Z$ and no descendent of v_i is in Z. A set $Z \subseteq V$ is said to *d-separate* $X \subseteq V$ from $Y \subseteq V$ if and only if Z d-separates every path from a node in X to a node in Y.

For example, in causal graph $G_1 = v_1 \leftarrow v_2 \to v_3$, set $\{v_2\}$ d-separates $\{v_1\}$ from $\{v_3\}$, while set $Z = \emptyset$ does not. In contrast, in causal graph $G_2 = v_4 \to v_5 \leftarrow v_6$, $\emptyset$ d-separates $\{v_1\}$ from $\{v_3\}$, while set $\{v_2\}$ does not. While *d-separation* is a property of sets of nodes in a causal graph, it has a direct link with conditional independence: if $X \subseteq V$ is d-separated from $Y \subseteq V$ given $Z \subseteq V$ in causal graph G, then $X \perp\!\!\!\perp Y \mid Z$ in every probability distribution that satisfies the causal Markov condition with respect to G. For example, in example

[3] By symmetry, also $P(Y \mid X, Z) = P(Y \mid Z)$

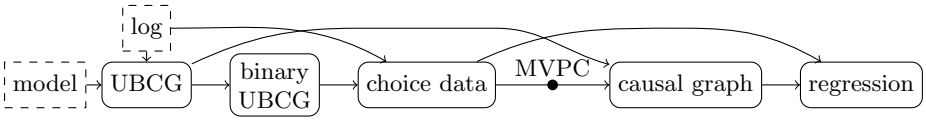


Figure 3: Overview of our method.

causal graph G_1, imagine that v_2 represents age, while v_1 and v_3 respectively represent the presence of arthritis and cardiovascular disease, two common age-related diseases. Now, $\{v_2\}$ d-separates $\{v_1\}$ from $\{v_3\}$, which means that in all probability distributions that satisfy the causal Markov condition w.r.t. G_1 it must be the case that the presence of arthritis and of cardiovascular disease are independent once the age is known.

Causal graphs that imply the same set of conditional independence relations, i.e., graphs that have the same set of d-separation properties, are *Markov equivalent*, and the set of all Markov equivalent causal graphs is a *Markov equivalence class* (MEC). In the case of two variables, graphs $v_1 \to v_2$ and $v_1 \leftarrow v_2$ are Markov equivalent, as the set of condition independence relations that both graphs imply is $\emptyset$. Markov equivalent causal graphs cannot be distinguished purely based on observational data. *Causal discovery* algorithms aim to reconstruct the causal graph from observational data, however these algorithms can thus merely identify the correct MEC of possible causal graphs [18].

5 Our Method

Our method consists of five steps, illustrated in fig. 3, that combine the information available in a process model and an event log. First, we obtain an *upper bound* on the causal graph (UBCG) that consists of all causal edges that are possible according to the process model. Second, we extract the choices made in the traces of the event log to obtain a data set of made choices (section 5.2). Third, we apply a causal discovery technique to obtain the MEC that is consistent with this choice data set. Fourth, in section 5.3 we combine the MEC with the UBCG, i.e., we shrink the MEC to contain only causal graphs that do not contradict the process model. Often, this MEC is a single causal graph. Finally, we estimate the size of the causal effects using this causal graph and a standard application of regression with backdoor adjustment [18].

5.1 Upper-Bound Causal Graphs

Given a process model, an *upper-bound causal graph* (UBCG) is a causal graph that contains all causal edges that do not violate the model. In this section, we describe how a UBCG can be computed for any DFM or process tree. The nodes of a causal graph indicate choices in the model. As causal graphs conceptually do not support loops – every choice in a causal graph can be made at most once –, if a model-choice is encountered multiple times in a trace, it must be represented multiple times in a causal graph. Thus, while the nodes of causal graphs differ slightly between DFMs and process trees, they both contain unfolding identifiers.

Directly Follows Models.

For DFMs, we use both the DFM and an event log for the construction of the UBCG. In constructing the UBCG, for each event it must be decided whether the event enters the "next" unfolding. We aim to minimise the number of unfoldings, as to minimise the number of nodes in the causal graph and to maximise the amount of information per node.

Intuitively, our starting point is the set of not-unfolded choices in the model. Our strategy is to create a total order of these choices: whenever the sequence of choices in a trace in the log goes backwards in this total order, we enter a new unfolding. Thus, our aim is to create a total order that minimises the total number of such backward steps.

We first create a *non-unfolded-choice graph* (nucg). To this end, each trace t of a log L is transformed using a function choicesTrace that takes a trace and a model, and returns the sequence of choices in the trace corresponding to the model. Such a function could be implemented using alignments [1]. Then, an edge is added between every pair of encountered options:

$$\text{nucg}(L, M) = [(o_i, o_j) \mid \langle \ldots o_i \ldots o_j \ldots \rangle = \text{choicesTrace}(t, M) \wedge t \in L] \tag{1}$$

For instance, fig. 4a shows the nucg of the DFM of fig. 2b.

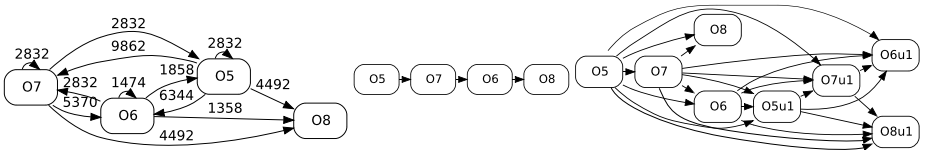
Next, we create a total order of choices (nucto) by repeatedly greedily adding choices with the least incoming edges: ((2))

$$\begin{aligned}
\text{nucto}(L, M) &= \text{nucto}'(\text{nucg}(L, M), 1, \{o \mid o \in \text{choicesTrace}(t, M) \wedge t \in L\}) \\
\text{nucto}'(G, r, O) &= \begin{cases} \{o \to r\} & \text{if } |O| = 1 \\ \text{nucto}'(G, r+1, O \setminus \{o\}) \cup \{o \to r\} & \text{otherwise} \end{cases} \\
\text{with } o &= \operatorname*{argmin}_{o \in O} \sum_{o' \in O \wedge o \neq o'} G((o', o))
\end{aligned} \tag{2}$$

For instance, fig. 4b shows the total order for our example of fig. 2b.

Then, the UBCG function ubcg takes a log and a model and returns a set of edges between choices. Intuitively, in a UBCG there is an edge between two choices if there is a potential causal relation between the source and target of the edge. In the context of a DFM, two choices can only have influenced one another if they appeared consecutively in a trace in the log. Thus, we add all such edges. To avoid cycles, whenever an edge goes *backwards* in the total order nucto, we increase the unfolding identifier u (3).

$$\begin{aligned}
\text{ubcg}(L, M) &= \bigcup_{t \in L} \text{ubcg}'(\langle\,\rangle, \text{choicesTrace}(t), \text{nucto}(L, M), 1) \\
\text{ubcg}'(t_c, \langle\,\rangle, R, u) &= \emptyset \\
\text{ubcg}'(\langle\,\rangle, \langle o \rangle \cdot t_o, R, u) &= \text{ubcg}'(\langle (o, \langle u \rangle) \rangle, t_o, R, u) \\
\text{ubcg}'(t_c \cdot \langle c \rangle, \langle o \rangle \cdot t_o, R, u) &= \{(c', c'') \mid c' \in t_c \cdot \langle c \rangle\} \cup \text{ubcg}'(t_c \cdot \langle c, c'' \rangle, t_o, R, u') \\
\text{with } c'' &= (o, \langle u' \rangle) \\
\text{and } u' &= \begin{cases} u & \text{if } R(o_c) < R(o), c = (o_c, t_x) \\ u+1 & \text{otherwise} \end{cases}
\end{aligned} \tag{3}$$



(a) Non-unfolded choice graph. (b) Total order. (c) UBCG ($k = 2$).

Figure 4: Constructing a UBCG from the DFM in fig. 2b.

To limit the size of the UBCG, the unfoldings can be maximised using a parameter k, which stops processing a trace if it reaches k unfoldings – processing the remainder of the trace remains future work.

In our example, the UBCG truncated to 2 unfoldings is shown in fig. 4c.

Process Trees. For process trees, it is trivial to decide when the "next" unfolding starts – with each execution of the second child of a loop –, thus we can construct the UBCG directly.

First, we describe the choices in a process tree recursively. • The silent and activity nodes τ and $a \in \Sigma$ do not possess any choices. (4) • The sequence, concurrent and interleaved operators do not add any choices (5). • The exclusive choice operator adds a choice between its children (6). • For each execution of an inclusive choice node, at least one child must be executed. Thus, there is one choice – which child to execute first – and another choice for each child – whether that child is executed as a non-first child (7). • For each execution of a loop node, a different choice is made – to proceed with the redo T_2 or to exit using T_3 (8). As loops can be arbitrarily nested, the unfolding identifier is a sequence of integers indicating the unfolding number of each encountered loop node. While a loop node thus describes a sequence of potentially infinitely many choices, every event log has finitely many finite traces. Therefore, we introduce a parameter k which indicates the times a loop node must be unfolded. In the following, I is initially $\langle\,\rangle$.

$$\mathrm{cs}(\tau, I) = \mathrm{cs}(a, I) = \emptyset \tag{4}$$

$$\mathrm{cs}(\oplus(T_1, \dots T_n), I) = \bigcup_{1 \le i \le n} \mathrm{cs}(T_i, I) \text{ for } \oplus \in \{\mapsto, \wedge, \leftrightarrow\} \tag{5}$$

$$\mathrm{cs}(\times(T_1, \dots T_n), I) = \bigcup_{1 \le i \le n} \mathrm{cs}(T_i, I) \cup \{(T_1, \dots T_n, I)\} \tag{6}$$

$$\mathrm{cs}(\vee(T_1, \dots T_n), I) = \{(\{T_l | 1 \le l \le n\}, I)\} \cup \{(T_i, \neg T_i, I) \mid 1 \le i \le n\} \tag{7}$$

$$\mathrm{cs}(\circlearrowleft(T_1, T_2, T_3), I) = \bigcup_{1 \le j \le k} \bigcup_{1 \le i \le 3} \mathrm{cs}(T_i, I \cdot \langle j \rangle) \cup \bigcup_{1 \le j \le k} \{(T_2, T_3, I \cdot \langle j \rangle)\} \tag{8}$$

Then, we can define the UBCG as a function ubcg, which produces a set of directed edges between (unfolded) choices of a process tree. • Intuitively, concurrency nodes do not induce any relation between their sub-trees: concurrent sub-trees are independent by definition, thus the choices in sub-trees cannot have causal relations with one another either. (10) • For exclusive choice, the choices made in its sub-trees depend on the choice made to select a sub-tree to execute.

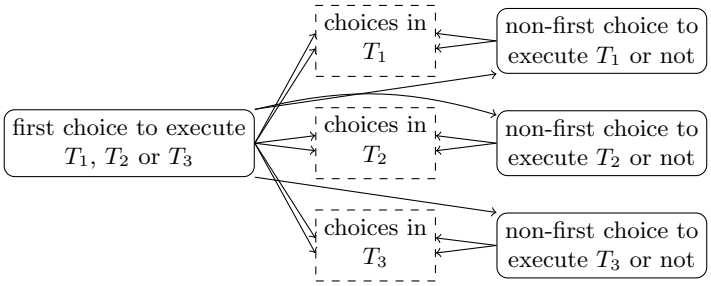


Figure 5: An UBCG of the execution of an inclusive choice $\vee(T_1, T_2, T_3)$.

As the sub-trees themselves are mutually exclusive, the choices in these sub-trees cannot have causal relations with one another. (11) • For sequence nodes, the choices in each sub-tree may causally depend on all choices in sub-trees before the current sub-tree. (12) Initially, I is $\langle\,\rangle$.

$$\text{ubcg}(\tau, I) = \text{ubcg}(a, I) = \emptyset \tag{9}$$

$$\text{ubcg}(\oplus(T_1, \dots T_n), I) = \bigcup_{1 \le i \le n} \text{ubcg}(T_i, I) \text{ for } \oplus \in \{\wedge, \leftrightarrow\} \tag{10}$$

$$\text{ubcg}(\times(T_1, \dots T_n), I) = \bigcup_{1 \le i \le n} \text{ubcg}(T_i, I) \cup (\{(T_1, \dots T_n, I)\} \times \text{cs}(T_i, I)) \tag{11}$$

$$\text{ubcg}(\mapsto(T_1, \dots T_n), I) = \bigcup_{1 \le i \le n} \text{ubcg}(T_i, I) \cup \bigcup_{1 \le i < j \le n} (\text{cs}(T_i, I) \times \text{cs}(T_j, I)) \tag{12}$$

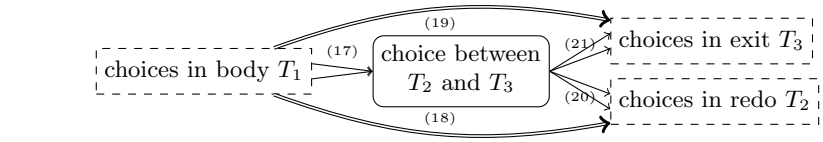
• To construct the UBCG of an inclusive choice node, we may assume that first a choice is made to execute a child, after which choices are made whether the remaining children are executed. Thus, all subsequent choices may depend on this first choice (14). Second, the choices within a child might also depend on the choice whether to execute that child (15). Figure 5 illustrates this for 3 children.

$$\text{ubcg}(\vee(T_1, \dots T_n), I) = \bigcup_{1 \le i \le n} \text{ubcg}(T_i, I) \cup \tag{13}$$

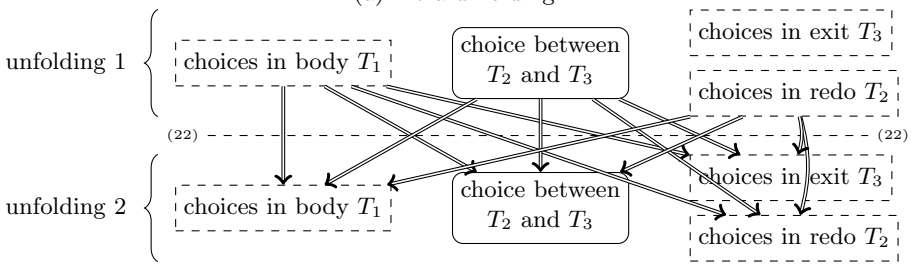
$$\{(\{T_1, \dots T_n\}, I)\} \times \bigcup_{1 \le i \le n} \{(\{T_i, \neg T_i\}, I)\} \cup \text{cs}(T_i, I) \cup \tag{14}$$

$$\bigcup_{1 \le i \le n} \{(T_i, \neg T_i, I)\} \times \text{cs}(T_i, I) \tag{15}$$

• The causal relations of a loop node consist of the relations of its sub-trees (16), relations within an unfolding of the loop (17)-(21) (see fig. 6a) and relations



(a) Intra-unfolding.



(b) Inter-unfolding.

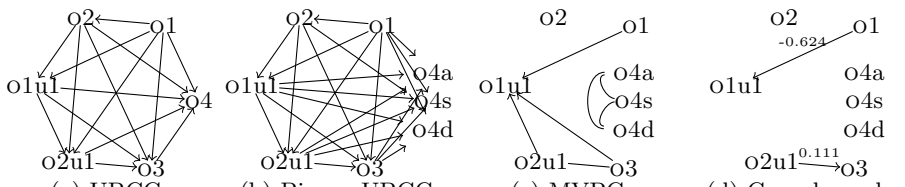
Figure 6: UBCG of $\circlearrowleft(T_1, T_2, T_3)$. Double edges indicate that every choice in the source of the edge is connected to every choice in the target of the edge.

between subsequent unfoldings (22) (see fig. 6b).

$$\begin{aligned}
\mathrm{ubcg}(\circlearrowleft(T_1,T_2,T_3), I) = {} & \bigcup_{1\le j\le k}\bigcup_{1\le i\le 3} \mathrm{ubcg}(T_i, I\cdot\langle j\rangle) \cup && (16)\\
& \bigcup_{1\le j\le k} (\mathrm{cs}(T_1, I\cdot\langle j\rangle) \times \{(T_2,T_3, I\cdot\langle j\rangle)\}) \cup && (17)\\
& (\mathrm{cs}(T_1, I\cdot\langle j\rangle) \times \mathrm{cs}(T_2, I\cdot\langle j\rangle)) \cup && (18)\\
& (\mathrm{cs}(T_1, I\cdot\langle j\rangle) \times \mathrm{cs}(T_3, I\cdot\langle j\rangle)) \cup && (19)\\
& (\{(T_2,T_3, I\cdot\langle j\rangle)\} \times \mathrm{cs}(T_2, I\cdot\langle j\rangle)) \cup && (20)\\
& (\{(T_2,T_3, I\cdot\langle j\rangle)\} \times \mathrm{cs}(T_3, I\cdot\langle j\rangle)) \cup && (21)\\
& \bigcup_{1\le j<j'\le k}\bigcup_{1\le i'\le 3} (\mathrm{cs}(T_1, I\cdot\langle j\rangle) \times \mathrm{cs}(T_{i'}, I\cdot\langle j'\rangle)) \cup\\
& (\mathrm{cs}(T_2, I\cdot\langle j\rangle) \times \mathrm{cs}(T_{i'}, I\cdot\langle j'\rangle)) \cup\\
& (\{(T_2,T_3, I\cdot\langle j\rangle)\} \times \mathrm{cs}(T_{i'}, I\cdot\langle j'\rangle)) && (22)
\end{aligned}$$

The parameter k can be chosen per node as to cover the longest unfolding of a loop in any trace of the log minus one, as the last unfolding always exits the loop. As some causal discovery techniques are exponential in the number of variables, a smaller k can be chosen. For example, fig. 7a shows the UBCG derived from the process tree in fig. 2a with maximum unfolding $k = 2$ for all nodes.

Binary UBCGs. The causal discovery technique we will use in section 5.3 requires binary choices. Therefore, we transform each n-ary choice into a set of binary choices using one-hot encoding in a post-processing step. That is, a choice between a, b and c is transformed into three choices: a vs. $\neg a$, b vs. $\neg b$ and c vs. $\neg c$. Figure 7b shows an example, where choice o4 has been split into



(a) UBCG. (b) Binary UBCG. (c) MVPC. (d) Causal graph.

Figure 7: Our method applied to a process tree (fig. 2a), with $k = 2$.

Table 2: Table of choice data of fig. 2a.

	o1	o2	o1u1	o2u1	o3	o4a	o4s	o4d
⟨o_sent, o_selected, o_created, o_accepted⟩	true	true	-	-	false	true	false	false
⟨o_selected, o_created, o_sent⟩	true	true	-	-	false	false	true	false

o4s(kip), o4a(ccepted) and o4d(eclined). The introduced binary choices have no causal relation.

5.2 Choice Data

The next step is to create a table of the decisions made in the event log. In this table, each row represents a trace, each column represents a choice from the model, and each cell indicates which option was chosen. As decisions depend on both model and log, we use alignments [1] to extract the decisions made for the choices in each trace according to the model. That is, steps in the log that have no equivalent in the model are ignored.

Certain choices are not encountered on certain paths through the process, resulting in missing data in the choice data set. For instance, the process tree model $\times(a, \mapsto(b, \times(c, d))$ contains two decision points, however the trace $\langle a \rangle$ yields a for the first choice, but the second choice is not encountered. Another source of missing data are loop executions. For instance, for the model in fig. 2a and the UBCG in fig. 7a, an example choice data table is shown in table 2

5.3 Causal Discovery

One of the oldest causal discovery algorithms is *PC* [25], which is able to identify the MEC under the assumption that there are no unobserved confounders. The PC algorithm starts from a fully connected undirected graph over all variables, and iteratively removes edges based on a series of statistical tests for conditional independence. The resulting graph summarises the MEC by keeping both directed and undirected edges, where the undirected edges mean that an edge between those nodes in either direction yields the same set of conditional independence relations and thus could not be distinguished.

Most causal discovery algorithms are unable to deal with missing values in the choice data. Therefore, we use the Missing Value PC (MVPC) algorithm [31], which is an extension of the PC algorithm that is able to handle missing data. If a value is missing, this is due to an earlier choice in the model, thus the cause

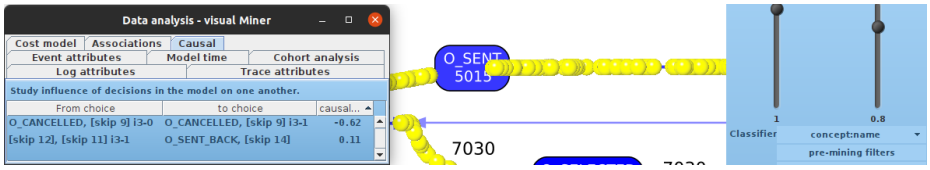


Figure 8: Implementation of causal analysis in the Visual Miner.

of missing data is fully observed (*Missing At Random* (MAR)). This is a step up from Missing Not At Random, for which MVPC is less precise.

We know from the UBCG that certain causal edges contradict the process model. Therefore, we filter these edges from the MEC (considering undirected edges as two directed edges). This modification makes the MVPC algorithm process-aware, i.e., it leverages domain knowledge from the process model to reduce the MEC of observationally equivalent causal graphs. This reduction generally yields a single causal graph since the UBCG contains no undirected edges.

Finally, we take the obtained causal graph and estimate the size of the causal effect of each the edge in the causal graph. We estimate these causal effect sizes using a regression with backdoor adjustment [18]. When estimating the causal effect for an edge $v_i \to v_j$ in the causal graph, the backdoor adjustment achieves d-separation between v_i and v_j for all paths between v_i and v_j other than the direct causal path $v_i \to v_j$. After this adjustment, the regression coefficient can be given causal interpretation as the *average treatment effect* (ATE).

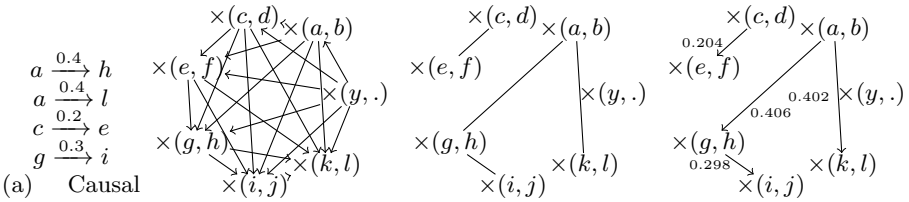
For example, fig. 7c shows the output of the MVPC step for our process tree of fig. 2a. Combining this graph with UBCG yields the causal graph in fig. 7d.

6 Evaluation

Implementation. Our method has been implemented as a prototype in the Visual Miner [15,16]: it shows the results of the analyses of this table in a tabular format, as shown in fig. 8. Intermediate steps are available to developers; future enhancements could include causal-graph editors and graph-based visualisations of identified causal dependencies.

Illustration: Synthetic Example. We apply our method to a synthetic process tree shown in fig. 9, and generate a log of 10 000 traces, while injecting the causal dependencies of fig. 9a. For instance, a choice for a in $\times(a, b)$ adds 0.4 to the probability of observing an h in $\times(g, h)$. This example contains several challenges: (i) *missing values* [by $\times(y, .)$]; (ii) *nested dependencies* [$\times(a, b)$ impacts $\times(g, h)$ and $\times(c, d)$ impacts $\times(e, f)$]; (iii) *indirect effects* [$\times(a, b)$ impacts $\times(g, h)$, which impacts $\times(i, j)$without direct effect]; and (iv) *direct effect after an indirect effect* [$\times(k, l)$ is yet again impacted by $\times(a, b)$]. Choosing a synthetic example with these edge cases demonstrates that our method can handle these challenges.

Then, we apply our method, yielding a UBCG (fig. 9b), the result of MVPC (fig. 9c), and the causal graph obtained by combining UBCG and MVPC (fig. 9d). Finally, ordinary least square regressions with backdoor adjustments for the true causal graph (fig. 9d) recovers estimates of the true causal effect s, up to sampling



(a) Causal effects. (b) UBCG. (c) MVPC result. (d) Causal graph.

Figure 9: Synthetic example $\times(y, \mapsto(\times(a, b), \times(c, d), \times(e, f), \times(g, h), \times(i, j), \times(k, l)))$.

Table 3: Results of the Applicability Experiment.

Log	miner	choices	UBCG edges	MVPC edges → (—)	causal graph edges	run time (s) mean±std.dev
BPIC12-a	DFMM	3	3	0 (6)	6	1.18±0.22
BPIC12-a	IMf	4	6	0 (4)	4	0.59±0.13
BPIC12-o	DFMM	42	594	7 (0)	4	1.29±0.23
BPIC12-o	IMf	16	120	1 (5)	7	3.80±0.29
Roadfines	DFMM	2	1	0 (0)	0	0.09±0.08
Roadfines	IMf	8	27	2 (3)	5	7.80±0.69
Sepsis	DFMM	288	23 776	not enough data for a particular edge		
Sepsis	IMf	148	5 485	0 (5)	5	74.52±3.63
BPIC17-o	DFMM	2	1	0 (0)	0	0.04±0.07
BPIC17-o	IMf	3	3	0 (4)	4	2.98±0.50

variation. Note that our method is successful in retrieving the structure of the causal graph, despite the challenges.

Demonstration: Illustrative Example. Continuing from section 2 and fig. 2a, our method identifies two causal relationships: the first occurrence of o1 skipping o_cancelled *causes* a reduction of the second occurrence of o1 executing o_cancelled by 0.624. Furthermore, executing the loop exactly two times (o2u1 exit) *causes* o_sent_back to be executed 0.111 more. This means that if the execution of o_sent_back is of concern (e.g., wasteful), the process can be optimised by reducing the number of times the loop is executed exactly two times. Notice that some of the associations identified in section 2 were in fact *not* causal.

Applicability: Real-Life Logs. To evaluate the practical applicability of our method, we applied it to several real-life event logs published by the IEEE Task Force on Process Mining, and models discovered by two miners: DFM Miner [16] (DFMM) and Inductive Miner - infrequent [14] (IMf). We measured the number of choices with log-bounded k, the edges in each of the graphs, and run time to apply MVPC (repeated 25 times on an i9-9980HK CPU with 32GB RAM). Table 3 shows the results; for one instance, the MVPC algorithm did not produce a result as one particular edge had a constant choice. In all cases, the run time of MVPC was in the order of a minute, which makes it faster than alignment computations. We conclude that it is practically feasible to run our method on real-life logs. Furthermore, at most 7 causal relations are identified, which seems promising as to not overload analysts.

Discussion. Our method assumes that there are no unobserved confounders for model decisions, which is a rather substantial assumption. While any technique would need to make this assumption, potential bias could be reduced by using more data from event logs. Future work could explore integrating trace or event attributes in the choice graph. For trace attributes, this would require to precisely determine the moment in time when the attribute value became known,

which might require domain expert knowledge. For causal analysis of choices, the moment of choice matters: the models $\mapsto(a, \times(b, c))$ and $\times(\mapsto(a, b), \mapsto(a, c))$ have the same language, but their moments of choice are different. In the first model a might have influenced the decision between b and c, but in the second model this is not possible. Event logs do not provide information about choices, so this cannot be verified using logs; our technique assumes that the given process model is correct.

7 Conclusion

Process mining aims to obtain insights from event logs, recorded from historic executions of business processes in organisations. In this paper, we introduced a causal method to study dependencies between choices within the control flow of process models. This method first derives an upper bound causal graph (UBCG) of choices in the model (for DFMs and process trees). Second, it transforms the log to a tabular data structure representing the choices made in each trace. Third, it applies an adjusted PC causal graph discovery algorithm, taking the UBCG into account. Fourth, it applies causal regression to find the causal relations. We illustrated the method and how it can be applied on an artificial and a real-life example, and applied the method in practice on several real-life event logs.

As future work, UBCGs for Petri nets could be constructed leveraging combined concepts of DFMs (markings/arbitrary loops) and process trees (concurrency). Furthermore, trace and event data can reveal additional confounding factors. Finally, the causal analysis could be summarised over unfoldings and projected on a model for easier analysis.

References

1. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-based fitness in conformance checking. In: ACSD. pp. 57–66. IEEE (2011)
2. Benjamini, Y., Hochberg, Y.: Controlling the false discovery rate: A practical and powerful approach to multiple testing. J. Royal Stat. Soc. **57**(1), 289–300 (1995)
3. Bozorgi, Z.D., Teinemaa, I., Dumas, M., Rosa, M.L., Polyvyanyy, A.: Process mining meets causal machine learning: Discovering causal rules from event logs. In: ICPM. pp. 129–136. IEEE (2020)
4. Bozorgi, Z.D., Teinemaa, I., Dumas, M., Rosa, M.L., Polyvyanyy, A.: Prescriptive process monitoring for cost-aware cycle time reduction. In: ICPM. pp. 96–103. IEEE (2021)
5. vanden Broucke, S.K.L.M., Weerdt, J.D.: Fodina: A robust and flexible heuristic process discovery technique. Decis. Support Syst. **100**, 109–118 (2017)
6. Brunk, J., et al.: Cause vs. effect in context-sensitive prediction of business process instances. Inf. Syst. **95**, 101635 (2021)
7. Choueiri, A.C., Portela Santos, E.A.: Discovery of path-attribute dependency in manufacturing environments: A process mining approach. JMS **61**, 54–65 (2021)
8. Geiger, D., Verma, T., Pearl, J.: Identifying independence in bayesian networks. Networks **20**(5), 507–534 (1990)
9. Günther, C.W., Rozinat, A.: Disco: Discover your processes. In: BPM Demos. vol. 940, pp. 40–44. CEUR-WS.org (2012)
10. Hompes, B.F.A., et al.: Discovering causal factors explaining business process performance variation. In: CAiSE. LNCS, vol. 10253, pp. 177–192 (2017)

11. Hsieh, C., Moreira, C., Ouyang, C.: Dice4el: Interpreting process predictions using a milestone-aware counterfactual approach. In: ICPM. pp. 88–95. IEEE (2021)
12. Kamal, I.M., Bae, H., Utama, N.I., Yulim, C.: Data pixelization for predicting completion time of events. Neurocomputing **374**, 64–76 (2020)
13. Leemans, S.J.J., Fahland, D.: Information-preserving abstractions of event data in process mining. Knowl. Inf. Syst. **62**(3), 1143–1197 (2020)
14. 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. LNBIP, vol. 171, pp. 66–78 (2013)
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Exploring processes and deviations. In: BPM Workshops. LNBIP, vol. 202, pp. 304–316 (2014)
16. Leemans, S.J.J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: Exploration & a case study. In: ICPM. pp. 25–32. IEEE (2019)
17. Narendra, T., et al.: Counterfactual reasoning for process optimization using structural causal models. In: BPM Forum. LNBIP, vol. 360, pp. 91–106 (2019)
18. Pearl, J.: Causality: Models, Reasoning, and Inference. Cambridge UP (2009)
19. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: EDOC. pp. 287–300. IEEE (2007)
20. Peters, S., et al.: Fast and accurate quantitative business process analysis using feature complete queueing models. Inf. Sys. **104**, 101892 (2022)
21. Qafari, M.S., van der Aalst, W.M.P.: Root cause analysis in process mining using structural equation models. In: BPM Workshops. LNBIP, vol. 397 (2020)
22. Qafari, M.S., van der Aalst, W.M.P.: Case level counterfactual reasoning in process mining. In: CAiSE Forum. LNBIP, vol. 424, pp. 55–63 (2021)
23. Qafari, M.S., van der Aalst, W.M.P.: Feature recommendation for structural equation model discovery in process mining. CoRR **abs/2108.07795** (2021)
24. Shoush, M., Dumas, M.: Prescriptive process monitoring under resource constraints: A causal inference approach. CoRR **abs/2109.02894** (2021)
25. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search, 2nd edition. MIT Press (2000)
26. Stierle, M.: Exploring Cause-Effect Relationships in Process Analytics - Design, Development and Evaluation of Comprehensible, Explainable and Context-Aware Techniques. Phd thesis, FAU Erlangen-Nürnberg (2021)
27. Sun, H., Liu, W., Qi, L., Ren, X., Du, Y.: An algorithm for mining indirect dependencies from loop-choice-driven loop structure via petri nets. IEEE TSMC (2021)
28. Sutrisnowati, R.A., Bae, H., Park, J., Ha, B.: Learning bayesian network from event logs using mutual information test. In: ICSOC. pp. 356–360. IEEE (2013)
29. Sutrisnowati, R.A., Bae, H., Song, M.: Bayesian network construction from event log for lateness analysis in port logistics. Comput. Ind. Eng. **89**, 53–66 (2015)
30. Tax, N., Teinemaa, I., van Zelst, S.J.: An interdisciplinary comparison of sequence modeling methods for next-element prediction. SoSyM **19**(6), 1345–1365 (2020)
31. Tu, R., Zhang, C., Ackermann, P., Mohan, K., Kjellström, H., Zhang, K.: Causal discovery in the presence of missing data. In: AISTATS. pp. 1762–1770 (2019)
32. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: CIDM. pp. 310–317. IEEE (2011)
33. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. **15**(2), 145–180 (2007)
34. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering workflow nets using integer linear programming. Computing **100**(5), 529–556 (2018)