Discovering and Analyzing Contextual Behavioral Patterns from Event Logs

Mehdi Acheli, Daniela Grigori, and Matthias Weidlich

Abstract—Event logs that are recorded by information systems provide a valuable starting point for the analysis of processes in various domains, reaching from healthcare, through logistics, to e-commerce. Specifically, behavioral patterns discovered from an event log enable operational insights, even in scenarios where process execution is rather unstructured and shows a large degree of variability. While such behavioral patterns capture frequently recurring episodes of a process' behavior, they are not limited to sequential behavior but include notions of concurrency and exclusive choices. Existing algorithms to discover behavioral patterns are context-agnostic, though. They neglect the context in which patterns are observed, which severely limits the granularity at which behavioral regularities are identified. In this paper, we therefore present an approach to discover contextual behavioral patterns. Contextual patterns may be frequent solely in a certain partition of the event log, which enables fine-granular insights into the aspects that influence the conduct of a process. Moreover, we show how to analyze the discovered contextual behavioral patterns in terms of causal relations between context information and the patterns, as well as correlations between the patterns themselves. Experiments with real-world event logs demonstrate the effectiveness of our techniques in obtaining fine-granular process insights.

Index Terms—Behavioral Patterns, Process Discovery, Pattern Mining, Contextual Data, Causality and Correlation.

1 Introduction

THE field of process mining develops techniques for a data-driven analysis of processes in domains such as healthcare, logistics, and e-commerce [1]. To this end, event logs that are recorded by information systems during process execution are exploited as an objective basis for process analysis, that is not biased by human perception. Event logs comprise traces, which are sequences of events, all of them related to a single execution of the process. An event, in turn, signals that a certain activity of the process was executed and also contains information on the context in which this happened. For instance, when considering the treatment process for a patient in a hospital, events may signal treatment steps such as a check of vitals, a blood draw, or an infusion. In addition, events and traces contain context information, e.g., on the age and sex of the patient, the results of the vitals check, or a drug code used for an infusion.

An important branch of process mining is process discovery [4]. Given an event log, discovery algorithms construct a formal process model that generalizes the behavior represented by traces of the log. In recent years, a plethora of process discovery algorithms has been proposed. They vary in terms of the model imposed for event logs [34]; rely on different modeling languages as a target formalism, e.g., Petri-nets [37], process trees [19], or BPMN [11]; and adopt diverse strategies to handle noise and incompleteness of event logs, e.g., by balancing over-fitting and under-fitting of the resulting model [45] or by filtering noise [9].

- *M. Acheli, D. Grigori are with Univ. Paris-Dauphine, PSL University, CNRS UMR[7243], LAMSADE, 75016 Paris, France.*
  E-mail: mehdi.acheli@dauphine.fr
  daniela.grigori@dauphine.fr
- *M. Weidlich is with Humboldt-Universität zu Berlin, Germany*
  E-mail: matthias.weidlich@hu-berlin.de

Traditional process discovery aims at the construction of a formal model that captures *all* the behavior observed in the log. However, such an approach is not well-suited for relatively unstructured application scenarios, where process execution shows a large degree of variability [1]. Indeed the understandibility of such "spaghetti-like" process models is compromised due to their structural peculiarities, in particular, the high number of arcs and activities [25], [26]. In such scenarios, in turn, it was advocated to base discovery on a set of smaller models or behavioral patterns instead of a single, complex, end-to-end model [2], [41], [42].

In essence, these patterns can be seen as generalizations of common notions of frequent item sets [3] and frequent sequences [39]. While traditional approaches for sequential pattern mining [30] concern solely sequence dependencies, such behavioral patterns have richer semantics. They are defined by a tree structure, where leaf nodes denote types of events (i.e., the executed activities) and non-leaf nodes denote behavioral operators, such as sequencing, concurrency, or exclusive choice. Through discovery of these patterns, insights on the conduct of the process are derived in terms of frequently recurring episodes of process behavior.

Various algorithms to discover such rich behavioral patterns from event logs have been proposed recently [2], [41], [42]. However, all existing approaches derive patterns that are frequent over *all* traces in a log, regardless of the context in which the events have been recorded. As a consequence, discovery is limited to patterns at a relatively coarse granularity: Only patterns that are frequent over all possible execution contexts are detected.

Such a context-agnostic approach to pattern discovery constitutes a severe limitation in terms of the insights that may be gained about a process. Episodes of process behavior that are common for a specific execution context, but not frequent over all contexts remain undetected. As an example,

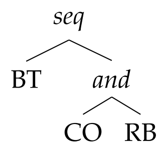
Trace ID	Context [Income, age]	Event Sequence
1	[low, <70]	EI ET PS ED BT BT GP TD SW CO RB
2	[low, <70]	ET EI CV XS BT SW CS D RB CO
3	[low, <70]	CI PS CV I BT XS SW E CO RB I
4	[low, <70]	CI CV PS XS BT D SW CS GP RB CO
5	[low, <70]	CI PS EI ED I XS GP TD CV
6	[high, <70]]	EI ET PS ED BT GP TD CO RB
7	[high, <70]	ET EI CV XS BT CS D CO RB
8	[high, <70]	CI PS CV I BT XS E CO RB I
9	[high, <70]	CI CV PS XS BT D CS GP CO RB
10	[high, <70]	CI PS EI ED I XS GP TD CV
11	[low, 70+]	ET PS ED BT GP TD SW CO RB
12	[high, 70+]	CI PS EI ED I XS GP TD CV

SW: Meet with social worker, CO: Checkout, BT: Blood test, CI: Check-In, GP: Give prescription, CS: Recheck sec. number, EI: Emergency intubation, ET: Emergency transfusion, PS: Process sec. number, CV: Check Vitals, XS: X-ray Scan, TD: Temp. Diagnosis, D: Diagnosis, ED: Emergency defibril., I: Infusion, E: Echography, RB: Retrieve belongings

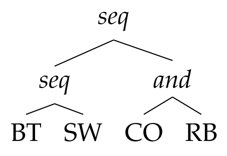
(a)

Sequential Patterns
BT RB
BT CO

(b)



(c)



(d)

Fig. 1: (a) Event log; (b) sequential patterns discovered with PrefixSPAN [30]; (c) behavioral pattern for the whole log; (d) behavioral pattern for the context [low, *]. All patterns have a support > 0.7 (i.e., occur in more than eight out of 12 traces).

consider the scenario given in Fig. 1. The traces of the event log contain events of treatment steps that patients follow in a hospital. There are two attributes representing contextual information: The income level of the patient (low or high) and the age group (<70 or 70+). For this log, traditional sequential patterns as derived, for instance, by PrefixSpan [30] (relative support > 0.7), provide limited insights as they fail to detect frequent concurrent behavior. This limitation is overcome by dedicated approaches to pattern discovery, such as [2], which would identify that events of type *BT* are frequently followed by the concurrent appearance of events of types *CO* and *RB*. However, either way, more specific patterns that are frequent within a particular context of process execution are not identified. For example, considering solely low income patients, there is a pattern of *BT*, followed by *SW*, which is again followed by the concurrent appearance of *CO* and *RB*. Detecting this contextual pattern provides insights (i.e., low income patients show a tendency to discuss the refund policy with a social worker) that would go unnoticed when neglecting context information. As such, contextual patterns enable fine-granular analysis of the correlations of contextual factors and process execution, thereby providing the basis for an exploration and assessment of causal effects.

In this paper, for the first time, we address the problem of discovering behavioural patterns while taking into account the dimensions induced by contextual data. Specifically, our contributions are summarized as follows:

(1) We introduce a model for contextual behavioral patterns. It includes different notions to link a behavioral pattern with contextual information of traces, such as contextual frequency, generality, and exclusiveness.
(2) We define the problem of discovering contextual behavioral patterns from an event log and propose an algorithm to solve it. . Based on a state-of-the-art algorithm for context-agnostic discovery of behavioural patterns, we show how to incorporate an exploration of possible contexts.
(3) We provide a methodology to analyze the discovered contextual behavioral patterns. We show how to verify causal relations between context information and patterns, as well as correlations between discovered patterns.

We evaluate our approach with four real-world event logs. Our results show good performances and confirm the effectiveness of the method. Indeed, not only there was additional insight discovered about patterns returned by [2] but more fine-grained patterns missed by context-agnostic methods were recovered. Moreover, an in-depth study of the discovered patterns reveals relevant insights on causal relations between context data and the found patterns. Finally, useful results on the interplay of behavioral patterns and how they react to contexts were derived.

In the remainder, Section 2 first reviews related work. We then provide the necessary background for our work in Section 3. Section 4 introduces a model for contextual behavioral patterns and an algorithm for their discovery. We elaborate on the analysis of discovered patterns in Section 5. An experimental evaluation is presented in Section 6, before we conclude in Section 7.

2 Related Work

The discovery of behavioral patterns defined with respect to their frequency in an event log connects several research areas, including sequential pattern mining and process discovery. In this section, we review related algorithms from either area. Moreover, we discuss approaches that incorporate contextual information in log analysis and that aim at the discovery of causal relations in recorded data.

Sequential pattern mining. Given a database of sequences of data elements, sequential pattern mining aims at the extraction of frequently recurring (sub-)sequences of elements [13]. While there has been decades of research on pattern mining algorithms, most of them proceed incrementally. For instance, the widely established GSP algorithm [39] combines pairs of sequential patterns of length k to obtain patterns of length $k+1$. This principle is also adopted in our previous work on behavioral patterns to generate rich behavioral patterns that include concurrency and exclusiveness.

Moreover, various approaches to optimize the runtime behavior of sequential pattern mining have been proposed. For instance, the PrefixSPAN algorithm [30] presents the notion of a projected database to evaluate the pattern candidates on the minimal number of traces possible. Also, measures such as maximality have been proposed to guide the search for the most relevant patterns [13]. Such optimizations and measures may be lifted to behavioral patterns, as shown in our previous work [2].

Approaches to association rule mining [3] and sequential rule mining [14], [15] are also related as they strive for the discovery of relations and correlations among elements in a database. As we show later, the respective ideas are useful when aiming at an understanding of the interplay of contextual behavioral patterns.

Lastly, approaches to incorporate contextual data in procedures for sequential pattern mining have been proposed, which include a formal definition of contexts, a characterization of different notions of frequentiality with respect to context hierarchies (e.g., a proposed notion of generality and exclusiveness) and an algorithm to mine the sequential patterns [31]. We later argue that the results exploited by this algorithm can be adapted to the discovery of behavioral patterns.

Process discovery. Process discovery constructs models with rich semantics that feature not only sequential dependencies, but concurrency, exclusive choices, or repetitive behavior from event logs, i.e., collections of traces [4]. While most approaches focus on the discovery of a single end-to-end model, their application in scenarios with relatively unstructured behavior will not yield useful insights [1].

To cope with event logs that show a large degree of variability in the behavior, it was suggested to first employ trace clustering before process discovery. Such clustering algorithms [7], [8], [16], [38], [40], group traces into homogeneous clusters. Applying process discovery to each cluster then enables the derivation of comparatively structured models. Such techniques are well-suited if a log contains few groups of similar traces. Yet, they do not cater for scenarios, where a partitioning of a log into a few groups is not possible.

Instead of trying to generalize all behavior, some discovery algorithms also rely on rule-based formalisms as a target domain. For instance, the Declare Miner [23] extracts rules that are satisfied by a certain share of traces based on a predefined set of templates. These rules come in the form of Linear Temporal Logic formulas and capture, presence, absence, ordering, and exclusiveness dependencies between types of events. Each rule, however, is limited to a relation between at most two event types, whereas our behavioral patterns link an arbitrary number of event types and also consider the context in which they are observed. Similarly, the Episode Miner [18] discovers frequent patterns in terms of partial orders over event types. The method, however, does not support loops and exclusive choices and, again, does not incorporate contextual information.

The notion of behavioral patterns used in our work [2] has first been proposed in [42] under the term Local Process Models (LPMs). However, first discovery algorithms [42] limited the size of patterns and were not grounded in the traditional definition of support, as known from sequential pattern mining. In addition, by following a generate-and-test approach, the respective algorithm incurred the overhead of redundant testing of pattern candidates, and suffers from high runtimes due to the evaluation of the patterns on the entire log. We addressed these aspects in our previous work that introduced the COBPAM algorithm [2]. It guarantees desirable properties for the discovered patterns (maximality and compactness) and implements pruning strategies, so that pattern candidates are explored at most once. To increase effectiveness of behavioral patterns, it was suggested to incorporate measures of utility and constraint satisfaction in the discovery process [41]. However, such extensions are orthogonal to our work: They prioritize and filter patterns that are frequent over *all* execution contexts, whereas we strive for the discovery of additional patterns that are frequent solely in particular contexts.

Contextual log analysis. Some approaches to incorporate contextual data in process discovery have been proposed. For instance, the rules discovered by the Declare Miner may be enriched with data attributes [24], which serve as conditions on the satisfaction of a rule. While in [24], such conditions are applied to the antecedent of rule, target conditions assigned to the consequent of a rule have bee introduced in [33]. Discovery of such conditions is assumed to rely on user-defined queries, though. Similar approaches have been proposed in [20] and [6] that strive for the discovery of event correlations. However, these methods utilize event data not trace attributes which makes them orthogonal to our work. Besides, our focus is not on an enhancement of discovered rules with data conditions, but on discovery of novel patterns that would be missed by a non-data-aware discovery approach.

To cope with contextual data in flexible processes, a process cube representation of event logs has been proposed [5], similar to what is known for OLAP [43]. Various views can be derived from a cube, which then serve as input to process mining algorithms. Our work, however, defines many types of frequencies based on a hierarchy on data attributes, and discovers correlations and dependencies between behavioral patterns and context data.

Moreover, data attributes have been incorporated in discovery of finite state machines [22], which neglects information on concurrency. Annotations of process models with data conditions may be derived by the Decision Miner [32] for branching points, and the context-aware Inductive Miner [35] for whole sub-parts of a model. All these approaches target traditional process models that generalize all seen behavior. As with the Declarative Miner family methods, the contextual data considered is that of events.

Discovery of causal structures. Discovery of causal relationships between data elements is a well-explored research area. One main branch is probabilistic causality, which relies on the discovery of Bayesian networks [28], or similar probabilistic formalisms. While many algorithms use Bayesian networks for the discovery of causal structures [17], [27], they are known to impose computational challenges. Some methods [12], [36] aim at increased efficiency by constraining the search to some local structures instead of discovering the whole network. Yet, this comes with limitations in terms of the result quality. In our work, we deal with observational correlated data. That is, we take up ideas on causal association rules discovery [21], due to their algorithmic efficiency.

3 Background on Behavioral Patterns

We discuss event logs and process trees to model behavioral patterns (Section 3.1), before turning to the related discovery problem and the COBPAM algorithm (Section 3.2).

3.1 Event Logs and Process Trees

We start by introducing event logs. Let A be a set of *activities*, and A^* the set of all sequences over A. As usual, for a sequence $\sigma_1 = \langle a_1, \ldots, a_n \rangle \in A^*$, we write $|\sigma_1| = n$ for its length, $\sigma_1(i) = a_i$, $1 \leq i \leq n$, for the i-th activity, and $\sigma_1.\sigma_2 = \langle a_1, \ldots, a_n, b_1, \ldots, b_m \rangle$ for the concatenation with a sequence $\sigma_2 = \langle b_1, \ldots, b_m \rangle \in A^*$. An interleaving of σ_1 and σ_2 is a sequence σ_3 of length $n+m$ that is induced by a bijection $\alpha : \{1, \ldots, n+m\} \to \{(s,j) \mid 1 \leq s \leq 2 \wedge 1 \leq j \leq |\sigma_s|\}$, which maps activities of σ_3 to those in σ_1 and σ_2, while preserving their original order in σ_1 and σ_2, i.e., for all $1 \leq i_1 < i_2 \leq n+m$ with $\alpha(i_1) = (s_1, k_1)$ and $\alpha(i_2) = (s_2, k_2)$, it holds that $\sigma_3(i_1) = \sigma_{s_1}(k_1)$, $\sigma_3(i_2) = \sigma_{s_2}(k_2)$, and $s_1 = s_2$ implies that $k_1 < k_2$. The set of all interleavings of σ_1 and σ_2 is denoted by $\{\sigma_1, \sigma_2\}_{\simeq}$.

A *trace* is a pair (id, σ) where $\sigma \in A^*$ is a finite sequence of activities and $id \in I$ is a trace identifier, with I being the set of possible identifiers. Then, an *event log* L is a set of traces, each having a unique identifier, i.e., $L \subseteq I \times A^*$ where $|\{id \mid (id, \sigma) \in L\}| = |L|$.

Given an event log, behavioral patterns may be discovered. In this work, we follow [42] and represent the behavioral patterns as process trees, a hierarchical model to define rich semantics over a set of activities. We recall the definition of a process tree [10]:

Definition 3.1. *A* process tree *is an ordered tree, where the leaf nodes represent activities and non-leaf nodes represent operators. Considering a set of activities A, a set of binary operators $\Omega = \{seq, and, loop, xor\}$, a process tree is recursively defined as:*
- *$a \in A$ is a process tree.*
- *considering an operator $x \in \Omega$ and two process trees P_1, P_2, $x(P_1, P_2)$ is a process tree having x as root, P_1 as left child, and P_2 as right child.*

The depth of a node (activity or operator) in the tree is the length of the path to its root. The depth of the tree is the maximal depth of any of its nodes.

The language $\Sigma(P)$ of a process tree P is a set of words, which is also defined recursively. For an atomic process tree $a \in A$, the language is $\Sigma(a) = \{\langle a \rangle\}$. For a process tree $x(P_1, P_2)$, in turn, the language is obtained by a function f_x that merges words of $\Sigma_1 = \Sigma(P_1)$ and $\Sigma_2 = \Sigma(P_2)$ depending on the semantics of the operator x. For the sequence and concurrency operator, the function concatenates and interleaves two words of either language, respectively:

$$f_{seq}(\Sigma_1, \Sigma_2) = \{w_1.w_2 \mid w_1 \in \Sigma_1 \wedge w_2 \in \Sigma_2\},$$

$$f_{and}(\Sigma_1, \Sigma_2) = \{w \mid w \in \{w_1, w_2\}_{\simeq} \wedge w_1 \in \Sigma_1 \wedge w_2 \in \Sigma_2\}.$$

For the exclusiveness operator, the languages are unified, while the language of the loop operator is obtained by alternating words of either language:

$$f_{xor}(\Sigma_1, \Sigma_2) = \Sigma_1 \cup \Sigma_2,$$

$$f_{loop}(\Sigma_1, \Sigma_2) = \{w_1.w_1'.w_2.\ldots.w_{n-1}'.w_n \mid n \in \mathbb{N}^*, \forall\, 1 \leq i \leq n : w_i \in \Sigma_1 \wedge w_i' \in \Sigma_2\}.$$

3.2 Discovery of Behavioral Patterns

Given an event log, only behavioral patterns that are useful from an analysis point of view shall be discovered. Following the reasoning presented in [42], we strive for patterns that are based on behavioral containment. For a trace (id, σ) of a log L, the behavior of a process tree P is exhibited by the trace, if there exists a word $w \in \Sigma(P)$ of the language of P, such that w is a projection of σ, i.e., the (order preserving) removal of activities of σ yields w, which we denote as $w \models \sigma$.

For example, in Fig. 1, the process tree in Fig. 1d defines the language $\{\langle BT, SW, CO, RB \rangle, \langle BT, SW, RB, CO \rangle\}$. Trace 1 of the event log in Fig. 1a exhibits this behavior, since $\langle BT, SW, CO, RB \rangle$, can be derived from trace 1 through projection. Note though that the second occurrence of BT is not part of the projection, as the language of the process tree does not define a respective repetition. Trace 8 is a counter-example. It does not exhibit the behavior since neither of the two words of the model can be derived by projection.

Frequent patterns. We capture the importance of patterns in terms of their frequency with the following measures:

Definition 3.2. *Given an event log L, the* count *of a process tree P is the number of traces that exhibit its behavior:*

$$count(P, L) = |\{(id, \sigma) \in L \mid \exists\, w \in \Sigma(P) : w \models \sigma\}|\,.$$

Its support *is the count over the size of the log:*

$$support(P, L) = \frac{count(P, L)}{|L|}.$$

Precise patterns. We are interested in precise patterns, based on the ratio of the behavior seen in the log and all the behavior allowed for by the pattern. The language of a process tree may be infinite due to loop operators. Thus, we exploit the n-language which is defined as the language of the model while traversing each loop at most n times:

Definition 3.3. *Given a fixed value of n for all trees and an event log L, the* precision *of a process tree P is defined as:*

$$precision(P, L) = \frac{|\{w \in \Sigma_n(P) \mid \exists\, (id, \sigma) \in L : w \models \sigma\}|}{|\Sigma_n(P)|}.$$

Here, a low n value allows for loops being 'precise' with little repetition, whereas a higher value imposes more repetitions before a model is considered to represent the log behavior well. This decision has to be taken manually by an analyst and offers control over the number of repetitions observed in the log that shall be taken as evidence for the presence of an actual loop in the model.

Compact patterns. To be useful for analysis, patterns shall also be compact. Given an event log L, a process tree P is *compact*, if it satisfies all following conditions:
- P does not exhibit the choice operator as a root node. If so, the process tree would be the union of separate trees, so that any support threshold may be passed through the aggregation of unrelated behavior.
- P does not include a choice operator, where, given L, just one of the children is frequent. The rationale is that adding further behavior to a frequent tree adds complexity, while the added behavior may not even appear in the log.

- P does not contain a loop operator $loop(P_1, P_2)$, where, given L, the behavior of P_2 does not appear frequently after P_1. While a word of the process tree may only show the behavior of P_1, the operator $loop(P_1, P_2)$ is not meaningful for traces showing only this behavior.

Combinations. COBPAM is a generate and test approach where k leaves trees are combined to get k+1 leaves trees. The trees are in fact partially ordered (an inclusion order where the results of the combination operation include the operands trees) and can be organized in a directed acyclic graph which we call construction graph. (Further details on the combination operation and the partial order we introduced can be found in [2]).

Maximal patterns. Behavioral patterns shall be maximal, to avoid the discovery of superfluous patterns that could be derived from other patterns. Indeed, a monotonocity property ensures that the combined trees are frequent whenever the combination results are so. As such, considering all patterns of at most depth i, a pattern is *maximal*, if it is not included in another pattern of depth smaller or equal to i.

Consider three trees *seq(a,b)*, *seq(a,c)*, and *seq(a,and(b,c))*. If all of them are frequent, the first two trees are not maximal since they can be infered from *seq(a,and(b,c))*. In our example in Fig. 1, we discover that a blood test (BT) is frequently followed by the checkout (CO) and the retrieval of belongings (RB), i.e., *seq(BT,and(CO,RB))*. Here, patterns *seq(BT,CO)* and *seq(BT,RB)* would not yield additional value. Moreover, we note that the pattern *seq(BT,and(CO,RB))* also provides richer information about the frequent behavior. Unlike the set of the two short patterns, *seq(BT,CO)* and *seq(BT,RB)*, it also states that the join occurrence of the checkout and the retrieval of belongings is frequent.

Pattern discovery with COBPAM. The COBPAM algorithm [2] for the discovery of behavioral patterns will serve as the foundation for our approach. Given an event log, it returns compact and maximal patterns that are frequent and have high precision. Intuitively, COBPAM explores the construction graph, starting from process trees of single activities. It discovers process trees that are built from frequent activities as well as frequent mergings of two infrequent activities through the choice operator. Each candidate tree is evaluated against a part of the log that may exhibit its behavior to calculate the support and precision. COBPAM further relies on a projection-based optimization and pruning rules (see [2] for details) to limit the number of process trees to evaluate and the number of traces used for evaluation. A maximum recursion depth d, which also limits the maximum depth for the discovered trees, can be used to explicitly terminate the discovery procedure.

4 Contextual Behavioral Patterns

This section presents our approach for taking into account contexts when discovering behavioral patterns. In order to obtain fine-granular insights into the behavior of a process, behavioral patterns can be associated to different contexts specified through conditions based on attributes that are attached to traces. This way, it is revealed whether a pattern is specific to a context, or materializes independently of any execution context.

4.1 Contexts

As a first step, we clarify the notion of a contextual event log. It includes attribute values that represent the context in which a trace was recorded. The possible attribute values are defined by a relation $\mathcal{R}(D_1, \ldots, D_n)$ with $D_i, i \in \{1, \ldots, n\}$, being the domain of the i-th attribute. In the remainder, we write d_i for the name of the i-th attribute. A tuple of this relation is assigned to a trace of the event log, as follows:

Definition 4.1. *Given a relation $\mathcal{R}(D_1, \ldots, D_n)$, a contextual event log is a pair (L, χ), where L is an event log and χ : a function that maps each trace (id, σ) of L to $\chi(id) = (v_1, \ldots, v_n)$, with $v_i \in D_i, \forall\, i \in \{1, \ldots, n\}$, i.e., a tuple of the relation $\mathcal{R}$.*

To define the notion of a context, we introduce D'_i as an extension of the domain D_i with a dedicated, unique symbol '$*$', which represents a wildcard. We capture this semantics by an inclusion order $\subset_{D'_i} = \{(v_i, *) \mid v_i \in D_i\}$, with $\subseteq_{D'_i} = \subset_{D'_i} \cup \{(v_i, v_i) \mid v_i \in D_i\}$ as its reflexive version, so that the wildcard symbol includes any value $v_i \in D_i$. The pair $(D'_i, \subset_{D'_i})$ defines an inclusion hierarchy $\mathcal{H}(d_i)$ on data attribute d_i. An example for two such hierarchies of our initial example is given in Fig. 2.

A context is defined as a tuple $(v_1, \ldots, v_n)$ with $v_i \in D'_i, \forall i \in \{1, 2, \ldots, n\}$. Contexts are organized through an inclusion order $\leq$, such that two contexts $C = (v_1, \ldots, v_n), C' = (v'_1, \ldots, v'_n)$ are ordered, denoted as $C \leq C'$, if $v_i \subseteq_{D'_i} v'_i\ \forall i \in \{1, 2, \ldots, n\}$. If $\exists 1 \leq i \leq n, v_i \subset_{D'_i} v'_i$, then context C' is said to be more general than C, while C is referred to as being more specific and we write $C < C'$ (strict inclusion order). A context $C = (v_1, \ldots, v_n)$ is atomic, if $v_i \in D_i, \forall i \in \{1, 2, \ldots, n\}$. We also designate a decomposition of a context C as the non-empty set of atomic contexts that are more specific than C.

For illustration purposes, consider the example of Fig. 2. The context (*,<70) is more general than (low, <70). As stated earlier, '$*$' is a wildcard symbol that represents any value v_i; meaning the context (*, <70) represents all the population with the age being smaller than 70, independent of the income. Contexts are derived directly from the data attribute values assigned to the traces in a log, and their hierarchy is well-defined due to the inclusion order over these values.

Note that the granularity of the contexts definition is controlled by the size of the domains of the data attributes and their number. In particular, for continuous data attributes, discretization may be employed, which divides the domain into several intervals, as to avoid generating a big number of contexts. The *age* attribute of our example illustrates such a discretization by considering solely two age groups (<70 and 70+) instead of the actual age values.

4.2 Contextual Behavioral Patterns

Now that we introduced contexts, we link a context to a contextual event log. That is, we consider the set of traces for which the contextual information is contained in the context.

Definition 4.2. *Let (L, χ) be a contextual event log and $C = (c_1, \ldots, c_n)$ a context. The associated event log of context C, is a contextual event log (L', χ') with $L' \subseteq L$, such that trace $(id, \sigma) \in L$ is part of L' if $\chi(id) = (v_1, \ldots, v_n)$ and for all v_i it holds that $v_i \subseteq_{D'_i} c_i$; and χ' is the restriction of χ to L'.*

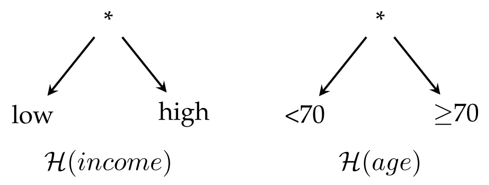


Fig. 2: Hierarchies on data attributes *income* and *age*

Moreover, we define the size of the context C with respect to (L, χ) as the size of the associated event log (L', χ').

A process tree is said to be **C-frequent** or frequent in C, if it is frequent in its associated log. In fact, we distinguish two types of behavioral patterns for a non-atomic context:

- **C-general behavioral patterns:** The patterns are frequent in C and in every descendant of C. Applied to our running example, these would be patterns answering questions such as what patterns are frequent in the low income population whatever their age?
- **C-exclusive behavioral patterns:** The patterns are C-frequent only in C and its descendants. In our example, these would be patterns answering the question what behavioral patterns are exclusively present for patients that are older than 70, whatever their level of income?

If a pattern is C-frequent in at least an atomic context and not C-General or C-exclusive in any context, it is called **AC-frequent**. If a pattern is C-frequent in the most general context, i.e., the associated event log is the original event log, it is called **log-frequent**.

4.3 Contextual Pattern Discovery Approach

Based on the notions of context and contextual behavioral patterns, we are ready to define the problem of discovering these patterns from a given event log. We first formulate the problem explicitly, before we discuss how to tackle it.

Problem 1 (Contextual Behavioral Pattern Discovery). *Given a contextual event log, (L, χ), the problem of* contextual behavioral pattern discovery *is to find the set Φ of C-general and C-exclusive behavioral patterns in each context of the hierarchy induced by the mapping function χ.*

The objective here is to exploit the data dimensions present in the event log, which have been ignored by previous methods for behavioral pattern discovery. To address the above problem, we introduce Contextual COBPAM (short CCOBPAM), as an adaptation of the COBPAM algorithm. Before defining the algorithm, we motivate the underlying design choices. Adopting the reasoning presented for sequential pattern mining with contexts [31], we observe the following: A behavioral pattern P is C-general, if and only if, it is frequent in the atomic contexts in the decomposition of C. In the same vein, a pattern P is C-exclusive, if and only if, it is frequent in the atomic contexts in the decomposition of C and not frequent in any other atomic context. Consequently, discovery of the two types of patterns, C-general and C-exclusive, in all contexts shall start with the discovery of behavioral patterns in atomic contexts.

The CCOBPAM algorithm takes as input the data attributes d_i that shall be considered for the definition of contexts, and a contextual event log. It returns a set of discovered behavioral patterns Φ, and three functions $atomLoc$, $genLoc$, and $excLoc$. These functions map each behavioral pattern P to, respectively, the atomic contexts, in which it is C-frequent; the contexts, in which it is C-general; and the contexts, in which it is C-exclusive.

Algorithm 1: CCOBPAM: Function *atomMine*

```
input  : $AC$, the set of atomic contexts;
         $\tau_S$, a support threshold; $\tau_L$, a precision threshold;
         $\tau_D$, a depth threshold.
output : $\Phi$, set of behavioral patterns;
         $atomLoc$.
1 for $C \in AC$ do
2     $\Gamma \leftarrow COBPAM(C, \tau_S, \tau_L, \tau_D)$, derive maximal compact
       C-frequent behavioral patterns;
3     for $P \in \Gamma$ do
4         add $P$ to $\Phi$;
5         add $C$ to $atomLoc(P)$;
```

CCOBPAM is based on two functions executed sequentially. The first one, *atomMine*, defined in Alg. 1, extracts the behavioral patterns from the atomic contexts by applying COBPAM. It returns the set of contextual behavioral patterns and, for each contextual pattern, the set of atomic contexts in which it is frequent (through $atomLoc$).

Such a pattern could be the one in Fig. 1d, noted Pex. It is discovered in both contexts [low, <70] and [low, 70+] resulting in $atomLoc(Pex) = \{$[low, <70], [low, 70+]$\}$

A second function, $nonAtomMine$, defined in Alg. 2, iterates over the non-atomic contexts to discover the C-general and C-exclusive patterns thus returning $genLoc$ and $excLoc$. We recall that a pattern P is C-general in a context C if P is C-frequent in each context of the decomposition of C and is C-exclusive if it is only frequent in that decomposition. Since *atomLoc(P)* points to the atomic contexts where it is frequent, then a C-general pattern in C is one such that the decomposition of C is part of $atomLoc(P)$ and a C-exclusive pattern in C is one such that the decomposition of C is equal to *atomLoc(P)*. Coming back to our running example, when executing $nonAtomMine$, for Pex, we iterate over the non-atomic contexts. Encountering [low, *], we realize that $atomLoc(Pex)$ is equal to the decomposition of [low, *]. As such, Pex is considered C-exclusive in [low, *] and $excLoc$ will be set to [low, *].

The support and precision of any pattern on a non atomic context NC is directly inferred from the aggregation of the counts and language seen in the decomposition of NC. Particularly, the support of a pattern in context NC is the sum of its counts in the atomic contexts from the decomposition of NC over the size of NC. The precision of a pattern in NC is the size of the union of the words seen in the atomic contexts of the decomposition of NC over the size of its language.

5 Pattern Analysis

In this section, we show how the result of discovering contextual behavioral patterns may be analysed to support the understanding of the considered process. To this end, we first discuss the identification of causal relations between contextual data and behavioral patterns. Second, we turn to the interplay of several contextual behavioral patterns.

Algorithm 2: CCOBPAM: Function *nonAtomMine*

```
input  : $NC$, the set of non atomic contexts;
         $\Phi$, set of behavioral patterns.
output: $genLoc$; $excLoc$.
1 for $P \in \Phi$ do
2     for $C \in NC$ do
3         if the decomposition of C is contained strictly in $atomLoc(P)$ then
4             add $C$ to $genLoc(P)$
5         else if the decomposition of C equals $atomLoc(P)$ then
6             set $excLoc(P)$ to $C$
```

Finally, we integrate these two analysis perspectives in a general methodology for pattern discovery and analysis.

5.1 Causal Relations between Data and Patterns

C-exclusive patterns are frequent solely in some context C and its descendants. This suggests that the frequency of the patterns is independent of the unconstrained attributes (i.e., set to '$*$') and is, in fact, correlated to the constrained ones. Knowing this, in order to interpret a pattern, it is useful to assess whether the traces actually suggest a causal relation between the context C and the occurrence of the pattern. Below, we limit ourselves to such causal relations for contexts that define a specific value for one of the attributes, while leaving the other attributes unconstrained. This restriction is motivated by the potential existence of causal relations between multiple constrained attributes, which would compromise the analysis. Specifically, if context C assigns a specific value v to some attribute d, the question is whether the occurrence of pattern P that is frequent in just context C and its descendants, is caused by the value v. The derivation of such a causal relation helps to interpret the discovered pattern and provides further insights into how the context influences the execution of the process.

We approach the analysis of causal relations between the context and a behavioral pattern by adopting the idea of a cohort study, as known in the medical domain. It aims to assess the impact of a risk factor, called the exposure variable, on an outcome variable. The procedure follows two groups with common characteristics apart from the risk factor. The group with the risk factor is the exposure group, the other one the non-exposure group. The common characteristics must be evenly distributed among the two groups and serve as controlled variables. A study may be a perspective study, if the groups are followed until the outcome appears, or a retrospective study, if it is conducted after the outcome has been observed as in our case.

In our context, a C-exclusive pattern P induces a certain correlation. With two variables for each trace, one indicating whether attribute d is set to value v and one indicating whether the behavior of P is exhibited, we may formalize the dependency as an association rule linking these variables. As a next step, we are interested in the presence of a causal association rule between the variables, which we assess following common procedures for cohort studies [21]. Causality is a stronger notion than correlation which states that a change in the exposure variable provokes a change in the outcome. However, for any point in time, concerning an individual in a population, we cannot observe the outcome in the presence *and* in the absence of the risk factor. The observed event is called factual and the hidden one is called counterfactual. To prove causality, the outcome should appear in the presence of the exposure and disappear otherwise (either if the presence of the exposure is the factual or the counterfactual). A cohort study works with observational data. For each data point in the exposure group, meaning an individual with some characteristics (controlled variables) where the risk factor is present, we simulate the counterfactual by choosing another data point with exactly the same characteristics in the non-exposure group. This concordance of the characteristics is, of course, an assumption of the method as some variables can be unrecorded. On another hand, if the outcome is clearly independent of some variable, there is no use to it as its value has no impact on the observed outcome. We can, in fact, use an individual with a different value for that variable as counterfactual because the outcome will not change.

In the following, we develop the method based on the above principles. A running example will illustrate each step. We suppose the existence of three attributes, d, a, b, in the contextual event log with two modalities for d (v and v') and three for each of a and b, $a_i, b_i, \forall 1 \leq i \leq 3$. The following steps ensue:

(1) We transform the relation $\mathcal{R}(D_1, D_2, D_3)$ that captures possible contexts in terms of attribute value combinations into a relation of Boolean variables $\mathcal{B}(B_1, \ldots, B_8)$. Here, the modalities of each attribute d, a, b are transformed into a set of Boolean indicator variables, od set to true (resp. od') if $d = v$ (resp. $d = v'$) and oa_i (resp. ob_i) set to true $\forall 1 \leq i \leq 3$ if $a = a_i$ (resp. $b = b_i$).
(2) The Boolean variable od that represents value v of attribute d is defined as the exposure variable.
(3) We define a Boolean outcome variable t per trace (id, σ) and pattern P that corresponds to $count(P, \{(id, \sigma)\})$ being one, i.e., it is true if the pattern is part of the trace, and false otherwise.
(4) Next, we identify among the attributes present in the contextual log, the ones that are correlated with the outcome variable, t and thus possible causal factors. These variables will serve as controlled variables. The reason is that we want to assess if, variable od being set to true, causes variable t being set to true among other possible causal factors. To this end, we apply the odds ratio as a measure of associativity, which should be significantly greater than one. For an association rule R $\rightarrow$ Q, the odds ratio is given as $(support(R \wedge Q) * support(\neg R \wedge \neg Q))/(support(\neg R \wedge Q) * support(R \wedge \neg Q))$ while the confidence interval's lower bound at 95% writes as in Eq. 1. If this latter value exceeds one then the association rule holds. For our example, we suppose that only oa_1, oa_2, oa_3, ob_1, ob_2 hold association rules with od.
(5) The event log is divided into an exposure group (traces where od is true) and non-exposure group (remaining traces). The groups are then filtered to ensure that the controlled variables are evenly distributed in both groups, in order to mitigate their effect. As it can be seen in Fig. 3, traces 6 and 11 were filtered out because both of their controlled variables could not be found in the opposite groups. By eliminating them, we ensure an equal distribution of the controlled variables values

$$exp(log(oddsRatio) - 1.96 * \sqrt{\frac{1}{supp(R \wedge Q)} + \frac{1}{supp(\neg R \wedge \neg Q)} + \frac{1}{supp(\neg R \wedge Q)} + \frac{1}{supp(R \wedge \neg Q)}}) \qquad (1)$$

Exposure group

Trace ID	od	oa_1	oa_2	oa_3	ob_1	ob_2	t
1	1	1	0	0	0	0	1
3	1	0	0	1	0	0	1
4	1	0	1	0	0	0	0
12	1	0	1	0	1	0	0
~~6~~	1	0	0	1	1	0	0

Non-exposure group

Trace ID	od	oa_1	oa_2	oa_3	ob_1	ob_2	t
9	0	1	0	0	0	0	0
2	0	0	0	1	0	0	1
8	0	0	1	0	0	0	1
5	0	1	1	0	1	0	0
~~11~~	0	0	1	0	0	1	0

Fig. 3: Exposure and non exposure groups construction.

between the groups.

(6) For each existing combination of values of the controlled variables, we assess the value of the outcome variable. Traces with a positive outcome in the exposure group and a negative one in the non-exposure group provide evidence for a causal relation (let their number be n_1). Traces with a negative outcome in the exposure group and a positive one in the non-exposure group, in turn, provide evidence against a causal relation (their number is noted n_2). If the ratio of the number of traces providing evidence for and those providing evidence against a causal relation, $\frac{n_1}{n_2}$, is larger than one, we conclude on the existence of the causal relation. The lower bound of the confidence interval of the latter ratio is given by $exp(log(\frac{n_1}{n_2}) - 1.96 * \sqrt{\frac{1}{n_1} + \frac{1}{n_2}})$. A value higher than 1 confirms the causal relationship [21].

Following this procedure, we are able to identify whether the context information of attribute d (or od, respectively) can indeed be seen as the cause of the occurrence of pattern P.

5.2 Interplay of Behavioral Patterns

Another perspective for the analysis of discovered patterns are correlations between the patterns themselves. Understanding the interplay between patterns, again, provides deeper insights into the conduct of the considered process.

In particular, we explore the interplay of behavioral patterns that are part of a frequent contextual behavioral pattern, which includes a sequence or concurrence operator as the root node. For such a setting, we investigate the presence of correlations between child patterns.

First, consider a discovered pattern $P = seq(P_1, P_2)$ with P_1, P_2 being subtrees. For such a pattern, we investigate whether there is evidence for an association rule $P_1 \rightarrow P_2$. From the existence of P, we know that the behavior of P_2 appears after the behavior of $P1$ during process execution. Two cases are possible; either P_2 is strongly associated with P_1, meaning P_2 is observed always and only when P_1 is observed; or it is loosely associated, if that is not the case. Again, we compute the odds ratio to evaluate the potential correlation. Fig. 4 gives an example of a process configuration where we can observe a case of interplay. It is to be noted that the computation of the odds ratio is done efficiently based on the results of the actual pattern discovery. Since the root of P is a sequence operator and due to monotonicity, $P1$ and $P2$ are also frequent and the set of traces in which they appear are already known.

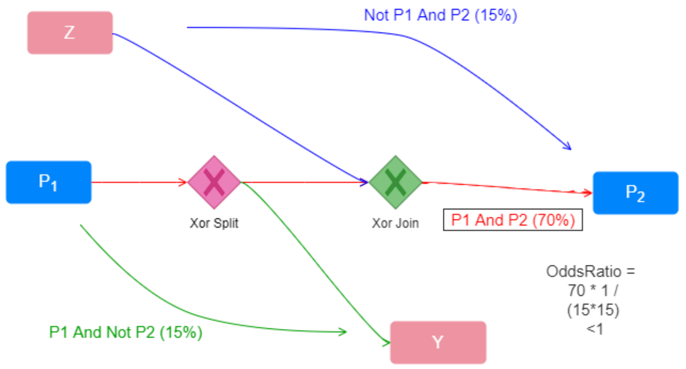


Fig. 4: Example of a non-valid behavioral rule.

The same procedure is also applied to behavioral patterns that have the concurrency operator as a root node, $P = and(P_1, P_2)$. The sole difference is that the semantics of the concurrency operator have to be incorporated: positive evidence for a correlation is provided by traces, in which the behavior of P_1 appears in parallel with the one of P_2, and never without it. Also, if the behavior of P_1 is not observed in a trace, neither is the behavior of P_2.

5.3 Methodology for Pattern Discovery and Analysis

Finally, we integrate the presented approach for the discovery of behavioral patterns with the procedures for the analysis of discovered patterns in a general methodology, as follows:

(1) To get a first overview of behavioral regularities, context-agnostic behavioral patterns are identified with the COBPAM algorithm, given a configuration of thresholds for the support and precision.
(2) Next, contextual attributes are selected, paying attention to their semantics and modalities (continuous or discrete). If needed, pre-processing is applied to adapt, normalize, or discretize the attribute values.
(3) A minimal context size is defined, in relation to the size of the log and, potentially, knowledge about the process under investigation. Contexts that don't meet the minimal size are discarded from the analysis and their associated traces deemed not representative enough of the context.

(4) Contextual behavioral patterns are mined with the CCOBPAM algorithm.
(5) Follow the interpretation guidelines, detailed below, to derive insights on the process.
(6) If suggested by these guidelines, adapt the support threshold, repeat the discovery of behavioral patterns, and note potential changes in the set of patterns.
(7) Finally, the presence of causal relations with context data, and correlations among patterns is evaluated for C-exclusive patterns according to the aforementioned procedures.

To complete the above methodology, we provide guidelines for the interpretation of discovered contextual behavioral patterns. CCOBPAM reveals patterns with respect to a hierarchy of contexts and considers different types of frequency that can be interpreted as follows:

- *AC-frequent*: The pattern is frequent in at least an atomic context and not C-General or C-exclusive somewhere else.
- *C-exclusive*: The pattern is exclusively frequent in the context and all its descendants, meaning that its occurrence is independent of the populations considered inside the context.
- *C-general*: The same interpretation as for a C-exclusive pattern applies. Yet, the pattern is also frequent somewhere else in the context hierarchy.

Moreover, patterns that are discovered by context-agnostic discovery with COBPAM *and* context-aware discovery with CCOBPAM are particularly interesting to derive insights on the process. In the following, we characterize the possible cases and provide an interpretation:

- *A log-frequent pattern appears as a C-exclusive pattern in the root context:* The pattern is not only frequent in the whole log, but also frequent in every context. This means that the pattern is strongly frequent, independent of the considered population.
- *A log-frequent pattern appears as C-exclusive or C-general in large contexts or C-frequent in large atomic contexts:* The pattern is actually frequent in some parts of the log that represent a significant portion of the log, which renders the pattern log-frequent. Yet, it is infrequent in other contexts. As such, the occurrence of a pattern that occurs very often (i.e., it is log-frequent) can be linked rather accurately to a specific context.
- *A log-frequent pattern appears as C-exclusive or C-General in only some small contexts or as C-frequent in some small atomic contexts:* The pattern is frequent in some parts of the log and close to frequent in other parts. This may point to a need to revisit the chosen support threshold.
- *A pattern is infrequent when the context is neglected, but is frequent under some context:* The identified behavior is frequent, but applies solely to specific contexts, showing the relevance of a contextual analysis.

6 Experimental Evaluation

This section presents an evaluation of our techniques for the discovery and analysis of contextual patterns.

6.1 Datasets and Setup

Our experiments used the following real-world event logs. They cover different domains and are publicly available.[1] Moreover, they are related to flexible processes, are of reasonable size to be explored and include data attributes with clear semantics.

- Sepsis: A log of a treatment process for Sepsis cases in a hospital. It contains 1050 traces with 15214 events that have been recorded for 16 activities.
- Traffic Fines: A log of an information system managing road traffic fines, containing 150370 traces, 561470 events, and 11 activities.
- WABO: A log of a building permit application process in the Netherlands. It contains 1434 traces with 8577 events, recorded for 27 activities.
- BPI_2019S: A 30% sample of the BPI Challenge 2019 log. The log belongs to a multinational company working in the area of coatings and paints and records the purchase order handling process. It regroups 479845 events distributed over 75519 traces with 41 event classes.

For the above event logs, we derived possible contexts based on the recorded attributes. Moreover, we considered a minimal size for each possible context, set to 50. Since in real-world data, the associated log of a certain context may not be representative of all possible behavior of the context population, such a minimal size helps to avoid the discovery of non-relevant behavioral patterns. The contexts considered for the event logs are summarized as follows.

- Sepsis: Two attributes were considered: 'InfectionSuspected', which is a Boolean variable stating if an infection is suspected, and 'Infusion', a Boolean variable stating if an infusion has been administered.
- Traffic Fines: Contexts were constructed from two attributes, 'amount', the amount of the fine, and 'VehicleClass', the type of the vehicle.
- WABO: From all data attributes available, we chose 'department' and 'channel' to construct contexts. The former represents the department working on the procedure. The latter refers to the channel of communication with the applicants.
- BPI_2019S: Each trace refers to a line item of a certain purchase order. The first attribute considered was the item category which specifies the method of invoice handling. The four categories are: '3-way matching, invoice after goods receipt', '3-way matching, invoice before goods receipt', '2-way matching (no goods receipt needed)', and 'Consignment'. The second attribute is the company concerned. We observed two values: 'companyID_0000' and 'companyID_0003'.

Our techniques have been implemented as a plugin in the ProM framework [44], a framework for process mining research (package *BehavioralPatternMining*). The discovery algorithm was configured with a threshold of 0.7 for the support and precision and two for the maximal depth of the patterns, unless stated otherwise.

To compare the efficiency of discovering behavioral patterns that are context-agnostic with the efficiency of the

1. https://data.4tu.nl/search?q=:keyword:"real%20life%20event%20logs"

TABLE 1: Run-times of CCOBPAM for different context definitions, including the numbers of atomic contexts

	Sepsis	Traffic Fines	WABO	BPI-2019S
LPM discovery	13m	Insuff.Mem	19m	>24h
0-attribute	49s (0)	3m (0)	8s (0)	3.4mn (0)
1-attribute	93s (2)	4.4h (49)	10s (1)	57mn (4)
1-attributeS	94s (2)	6.5mn (3)	12s (3)	5.3mn (2)
2-attribute	2m (3)	3.5h (66)	13s (3)	57mn (4)

proposed method for context-aware discovery, we consider the algorithm for LPM discovery [42], which is also available as a ProM plugin. It comes with a single parameter, which is a bound for the number of LPMs to discover. We set this bound to 500, the maximal possible value.

All experiments have been conducted on a PC (i7-2.2Ghz CPU, 16GB RAM) running Windows 10.

6.2 Efficiency of Patterns Discovery

We start by evaluating the influence of the introduction of contexts on the baseline algorithm COBPAM.

Table 1 reports a break-down of the run-times of the discovery of contextual patterns with CCOBPAM . For each log, we first list the run-time without incorporating context (0-attribute, corresponds to COBPAM on the whole log). Then, a 1-attribute experiment considers solely the first attribute mentioned for each event log in Section 6.1 while the 1-attribute experiment considers the second attribute mentioned. The 2-attribute experiments incorporated two attributes. The table also includes information on the number of atomic context, given in brackets after the run-time. This number of atomic contexts depends on the number of attributes considered and their domain. Lastly, the runtime of LPM on the whole log is included for comparison purposes (for a quantitative and qualitative comparison between COBPAM and LPM, see [2]).

First, we can observe that extending COBPAM to take into account contexts increases execution time. However, even with that additional overhead, LPM execution is still slower than CCOBPAM. Second, the change in run-time introduced by adding a new attribute in CCOBPAM can be explained by two opposite forces. Contexts are smaller when increasing the number of attributes, which, in general, reduces the required run-time On the other hand, when increasing the number of attributes, i.e., when increasing the number of atomic contexts, new construction graphs, along with the specific set of frequent trees they contain, are observed in each additional context. In some cases, these construction graphs are bigger or contain more frequent trees and need to be explored, which requires additional run-time. As can be seen in Table 1, both of these effects have varying impact, depending on the considered event log. The scale of the change in run-times depends on the number of atomic contexts added.

In the following, we will attempt to validate the previous statement. In Table 1, when increasing the number of considered attributes, the sizes of the atomic contexts explored changes. Accordingly, we used different samples (from 10% to 100%) of the Sepsis log, each simulating an atomic context size. In Table 2, we report the run-times along with further statistics on: *nbEvents*, the total number of events; *Q1, Q2, Q3*, the first, second, and third quartile on the size of the traces; *nbTrees*, the number of discovered trees; *nbEva*, the number of trees evaluated.

TABLE 2: Scalability analysis using the Sepsis event log

%	nbEvents	Q1	Q2	Q3	nbTrees	nbEva	Run-time (s)
10	1434	9	13	16	382	17916	9260
20	2892	8	13	16	287	8501	14
30	4416	8	13	17	476	38151	8289
40	5685	9	13	17	370	8692	27
50	7045	9	13	17	356	8727	31
60	8904	10	13	17	484	13645	85
70	10144	9	13	16	344	8624	40
80	11632	8	13	16	352	8022	37
90	12693	9	13	16	377	8093	41
100	15214	9	13	16	375	8029	49

TABLE 3: Run-times of CCOBPAM on logs w/o repetitive activities

	Sepsis	Traffic Fines	WABO	BPI_2019S
Run-time	61s	3.3h	12s	51mn
Relative run-time	50%	94%	92%	89%
R1	37%	1%	3%	18%
R2	52%	1%	2%	28%

The results indicate that the run-times are subject to large variability in comparison to the size of the log, which confirms that the log size (the first force discussed) is only one of many factors influencing the hardness of pattern discovery. Another important factor is the number of frequent patterns and, hence, the number of trees to evaluate in the search process (the second force mentioned). Since the evaluation of a tree is computationally hard (support and precision have an exponential run-time complexity in the size of the tree), the highest overall run-times are observed when the largest numbers of trees have to be evaluated. However, this is only one factor and a large number of evaluations may also turn out to not increase the run-time significantly. All the other factors are related to the construction graph and one of them is the number of loop-involving patterns which we will discuss in the next section.

Impact of Repetitive Activities We also explored the impact of repetitive activities, i.e., multiple events of the same activity in a trace, which may indicate a pattern containing a loop operator. Table 3 shows the run-times of executing the 2-attribute CCOBPAM, when removing all but the first of the events of a single activity per trace. Here, *R1* is the ratio of the number of removed events and the total number of events; *R2* is the mean ratio of the number of removed events and the trace size. The relative run-time with respect to the standard execution of CCOBPAM is also given.

There is a reduction in run-time, when the events of repetitive activities are removed. This is due to the overhead in the computation of support and precision for trees that contain loop operators.

6.3 Effectiveness of Pattern Discovery

Next, we conducted a quantitative analysis on the patterns returned by CCOBPAM for the WABO and BPI_2019S logs. For the other logs, without further domain knowledge, the

TABLE 4: Contexts statistics for WABO.

	Size	C-exclusive	AC-frequent	Rules
[General, Internet]	1211	/	8	9
[General, Post]	53	/	2	19
[General, desk]	105	/	2	9
[General, *]	1369	30	/	19
[*, *]	1369	30	/	19

TABLE 5: Contexts statistics for BPI_2019S.

	Size	C-exclusive	AC-frequent	Rules
[Consignment, comp.ID_0000]	4331	/	0	0
[inv. before GR, comp.ID_0000]	66318	/	41	20
[inv. after GR, comp.ID_0000]	4565	/	279	10
[2-way match, comp.ID_0003]	304	/	2	1
[Consignment, *]	4331	/	/	0
[invoice before GR, *]	66318	/	/	0
[*, companyID_0003]	304	/	/	0
[2-way match, *]	1369	/	/	0
[*, companyID_0000]	75214	1	/	0
[invoice after GR, *]	4565	/	/	0
[*, *]	75518	/	/	0

number of possible contexts and patterns turned out to be overwhelming. Our observations are summarized in Table 4 and Table 5. First, we analyze the contexts constructed with respect to the minimal context size in WABO. There are three atomic contexts, as shown in the hierarchy in Table 4. Overall, 30 patterns are C-exclusive in the root context. Some patterns are neither C-exclusive nor C-general. They are AC-frequent and their number is indicated in the corresponding column. Concerning BPI_2019S log, there are 4 atomic contexts and only one pattern is C-exclusive. All the other 319 patterns discovered are AC-frequent.

6.4 Patterns Analysis

6.4.1 *Causal Relations between Data and Patterns*

For the WABO event log, causal relations between context data and patterns, see Section 5.1, could not be found due to the peculiar hierarchy of contexts (the counterfactual of the constrained attribute in the unique context that can hold C-exclusive patterns is only present in discarded small-sized contexts). We, therefore, considered the Sepsis event log and explored causality of context data and patterns. Based on the two Boolean attributes 'InfectionSuspected' and 'Infusion', three contexts were constructed: *[true, true]*, *[true, false]* and *[false, false]*. The total number of patterns discovered was 968 and 77 of those were C-exclusive. We found causal relations for 35 of the C-exclusive patterns. Specifically, they were caused by the exposure variable $InfectionSuspected = true$, highlighting that a suspected infection can be seen as the root cause of various behavioral regularities like *seq(seq(ER Triage, Leucocytes), CRP)*. As for BPI_2019S, the only C-exclusive pattern did not show any causal dependencies while no C-exclusive patterns were accounted for in Traffic Fines.

6.4.2 *Interplay of Behavioral Patterns (Rules)*

The column *Rules* in Table 4 and Table 5 indicates how many association rules that describe the interplay of the discovered patterns were found in each context in WABO log and BPI_2019S respectively, see Section 5.2. Specifically, the number represents the number of patterns that hold a rule. An example of those patterns is Tree (1) in Fig. 5 which holds a rule in all contexts of the log. In fact, there is a strong dependency between *seq(T02, T04)* and *seq(T06, T10)* (child subtrees of the root) in all contexts, in which the tree is frequent. Note, however, that the presence of a rule in a context does not mean that the same rule is present also in a more general context. The opposite does not hold either. The existence of a rule is independent between contexts. This is known as the Simpson's Paradox [29]. We give the example of a process $P = seq(P_1, P_2)$, with P_1, P_2, two subtrees which is C-General in [low, *] while holding a rule in that context (as shown in Table 6). The pattern, however, does not satisfy a rule in the more specific contexts [low, <70] (see Table 7) and [low, 70+] (see Table 8). Discovering the occurrences of the Simpson's Paradox is another upside of the contextual technique. Uncovering statistical association that is not verified when stratifying the data prevents the analyst from considering misleading interplays. Indeed, the strong association between P_1 and P_2 in [low, *] is spurious as neither the "<70" nor the "70+" population confirms it. It is to be noted that when variables are missing, there is a risk of taking fallacious interplays for real ones. This happens when stratifying the data on the missing variables brings opposite results on the behavioral rules.

TABLE 6: Presence of P_1 and P_2 in the traces of [low, *], the odds ratio being 7.

	P_2	$\overline{P_2}$
P_1	70	10
$\overline{P_1}$	10	10

TABLE 7: Presence of P_1 and P_2 in the traces of [low, <70], the odds ratio being 0.93.

	P_2	$\overline{P_2}$
P_1	28	6
$\overline{P_1}$	5	1

6.4.3 *Methodology in practice: Interpretation of Patterns*

In this section, we apply our methodology to the patterns discovered. As the procedure needs to determine which patterns were discovered both by CCOBPAM (on the two attributes) and COBPAM (on the whole log) and in which contexts, manually, we could only apply it on WABO and BPI_2019S due to the limited number of patterns to inspect. Table 9 shows that CCOBPAM discovered 40 contextual patterns, whereas COBPAM discovered 33 context-agnostic patterns. All of these were also discovered by CCOBPAM, so that they are listed in the *Common* column. Of the common patterns, 30 are C-exclusive and three AC-frequent. So, these three patterns are frequent in the log. Yet, they are frequent

TABLE 8: Presence of P_1 and P_2 in the traces of [low, 70+], the odds ratio being 0.6.

	P_2	$\overline{P_2}$
P_1	42	10
$\overline{P_1}$	7	1

TABLE 9: Pattern statistics of CCOBPAM and COBPAM for WABO.

	CCOBPAM	Common
C-exclusive	/	30
AC-frequent	7	3

TABLE 10: Pattern statistics of CCOBPAM and COBPAM for BPI_2019S.

	CCOBPAM	Common
C-exclusive	1	0
AC-frequent	291	28

solely in some contexts, not in all of them. Seven patterns were revealed only by our novel CCOBPAM algorithm and are AC-frequent: They are frequent in some atomic contexts, but not in the whole log. Hence, they are missed by context-agnostic discovery. Table 10 shows on its turn that the only C-exclusive pattern discovered by CCOBPAM is not log-frequent. Moreover, among the 319 AC-frequent patterns returned, 28 were also log-frequent.

For a more concrete example on the contextual patterns analysis methodology, we illustrate in Fig. 5, a discovered pattern for each of the cases of pattern frequentiality presented in the end of Section 5.3. It is important to note that any tree discovered in WABO or BPI_2019S falls under one of the four categories we are going to exemplify. The trees we present here were selected randomly. Tree (1) is observed in all contexts. Since the tree is C-exclusive in the whole population, it is frequent regardless of the department and communication channel used in the building permit application process, thereby representing generic, recurring behavior. Tree (2), in turn, exemplifies a log-frequent behavior that occurs solely in two small atomic contexts, specific to applications handled by desks and through the post office. The total size of these contexts is only 158 traces, while the whole log contains 1434 traces. This indicates that the pattern is close to frequent in the other contexts suggesting a potential need to re-calibrating the support threshold.

Tree (3) is log-frequent and frequent in the largest atomic context concerning communication channel (i.e., the internet channel). This is another insight regarding the specific set of traces in which a pattern is frequent. Another interesting tree is (4), which is not discovered when neglecting contextual information, because it is only frequent in cases handled via the internet channel. As such, it exemplifies that context-aware discovery reveals patterns that are otherwise missed since they relate to a small set of traces.

6.5 Discussion and future extensions

The experimental evaluation can be extended in several ways to address its current limits:

- **Unavailability of contextual logs:** The scarcity of real life logs containing contextual data prevented us from doing more extensive experiments to study the influence of contexts. Even when data attributes are present in some logs, they are not documented and their meaning stays obscure. An industrial use case would allow us to do a more in-depth analysis.

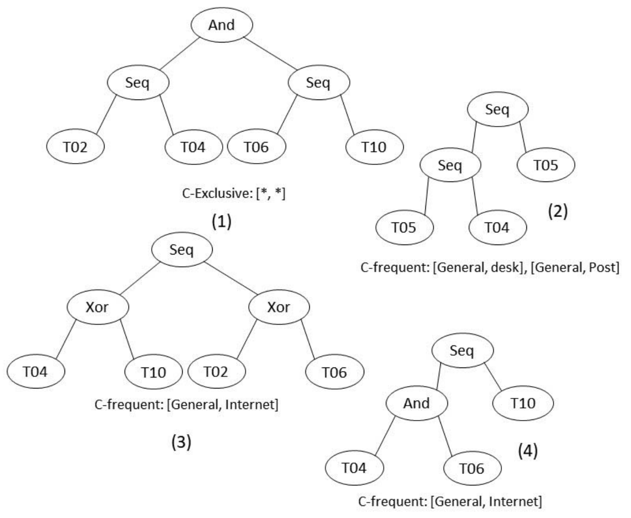

And
Seq
Seq
T02
T04
T06
T10
C-Exclusive: [*, *]
(1)
Seq
Seq
T05
T05
T04
(2)
C-frequent: [General, desk], [General, Post]
Seq
Xor
Xor
T04
T10
T02
T06
C-frequent: [General, Internet]
(3)
Seq
And
T10
T04
T06
(4)
C-frequent: [General, Internet]

Fig. 5: Illustrative examples of behavioral patterns mined with CCOBPAM.

- **Missing values for context attributes:** In our experiments, traces whose context attributes have no values, have not been included in the discovery. In a real application, the missing values could be handled using methods known from databases in order to infer their values. Alternatively, a Null value context could be defined and the discovered patterns in it could help in identifying causes of missing values.
- **User validation:** While the effectiveness evaluation showed that our method gives a more compact view of the behavior of a flexible process, a user-assisted evaluation should be conducted to conclude that business analysts find indeed a significant aid using our method.

Our approach for contextual behavioral pattern discovery and analysis can be extended in several directions that we plan to explore in future work:

- **Visualization techniques:** While COBPAM (and a fortiori CCOBPAM) outputs only compact and maximal patterns (compared with existing approaches that output redundant patterns), the number of patterns may be too large and user guidance and visualization techniques may be needed.
- **Relevant contexts definition**: Constructing suitable contexts that ensure actionable insights relies on the expert domain knowledge (e.g., deciding 70 as a boundary for the age related context). Possible extensions are defining an interface allowing users to easily define contexts and explore them or automatically identifying atomic contexts in which interesting patterns may appear.
- **Memory issues:** The method consumes a lot of memory as all previously discovered patterns need to be maintained throughout the execution of the algorithm. Because of this, our techniques fail in handling large event logs.
- **Data exploitation: Our methods use data associated to traces and ignores data attributes specified on events. Other methods specialize on those specific attributes** [6], [20], [24], [33]. Incorporating the whole

data dimension present in the log in the pattern analysis can help bring additional insights.

7 Conclusion

In this paper, we proposed the notion of contextual behavioral patterns, an algorithm for extracting them from an event log generated by a flexible process, and a methodology for analyzing the discovered patterns. Contextual behavioral patterns enable fine-granular insights in how a process is conducted, by not only pointing to recurring behavior, but revealing the contextual factors under which the behavior is observed. This way, process analysis becomes (i) more exhaustive by including patterns that are not frequent for the whole log, but solely for a certain context; and (ii) more precise as the impact of context is made explicit. For efficient discovery of contextual behavioral patterns, we presented CCOBPAM, an extension of the COBPAM algorithm. Moreover, our methodology for pattern analysis further supports the interpretation of the discovered patterns. In particular, we showed how to explore causal relations between context data and patterns, as well as the interplay of patterns in terms of correlations.

An experimental evaluation with real-world event logs demonstrated how our approach improves over the state-of-the-art in behavioral pattern discovery in terms of relevance of extracted patterns and resulting insights. Our results on the efficiency of discovery also illustrate general feasibility.

References

[1] Van der Aalst, W.: Process mining: Data science in action. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

[2] Acheli, M., Grigori, D., Weidlich, M.: Efficient discovery of compact maximal behavioral patterns from event logs. In: Giorgini, P., Weber, B. (eds.) Advanced Information Systems Engineering - 31st International Conference, CAiSE 2019, Rome, Italy, June 3-7, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11483, pp. 579–594. Springer (2019). https://doi.org/10.1007/978-3-030-21290-2_36, https://doi.org/10.1007/978-3-030-21290-2_36

[3] Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93. pp. 207–216. Association for Computing Machinery (ACM), New York, New York, USA (1993). https://doi.org/10.1145/170035.170072, http://portal.acm.org/citation.cfm?doid=170035.170072

[4] Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. IEEE Transactions on Knowledge and Data Engineering **31**(4), 686–705 (2018)

[5] Bolt, A., Van Der Aalst, W.M.: Multidimensional process mining using process cubes. In: Lecture Notes in Business Information Processing. vol. 214, pp. 102–116. Springer Verlag (2015). https://doi.org/10.1007/978-3-319-19237-6_7

[6] Bose, R.P.C., Maggi, F.M., Van Der Aalst, W.M.: Enhancing declare maps based on event correlations. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 8094 LNCS, pp. 97–112. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40176-3_9

[7] Bose, R.P.C., Van Der Aalst, W.M.: Trace clustering based on conserved patterns: Towards achieving better process models. In: LNBIP (2010)

[8] Bose, R.J.C., Van der Aalst, W.M.: Context aware trace clustering: Towards improving process mining results. In: Proceedings of the 2009 SIAM International Conference on Data Mining. pp. 401–412. SIAM (2009)

[9] vanden Broucke, S.K., De Weerdt, J.: Fodina: A robust and flexible heuristic process discovery technique. Decision Support Systems **100**, 109–118 (8 2017)

[10] Buijs, J.C., Van Dongen, B.F., Van Der Aalst, W.M.: A genetic algorithm for discovering process trees. In: CEC 2012. pp. 1–8. IEEE (6 2012)

[11] Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: BPMN Miner: Automated discovery of BPMN process models with hierarchical structure. Information Systems **56**, 284–303 (3 2016)

[12] Cooper, G.F.: A simple constraint-based algorithm for efficiently mining observational databases for causal relationships. Data Mining and Knowledge Discovery **1**(2), 203–224 (1997). https://doi.org/10.1023/A:1009787925236

[13] Fournier-Viger, P., Chun, J., Lin, W., Kiran, R.U., Koh, Y.S., Thomas, R.: A Survey of Sequential Pattern Mining. Ubiquitous International **1**(1), 54–77 (2017)

[14] Fournier-Viger, P., Gueniche, T., Zida, S., Tseng, V.S.: Erminer: sequential rule mining using equivalence classes. In: International Symposium on Intelligent Data Analysis. pp. 108–119. Springer (2014)

[15] Fournier-Viger, P., Wu, C.W., Tseng, V.S., Cao, L., Nkambou, R.: Mining Partially-Ordered Sequential Rules Common to Multiple Sequences. IEEE Transactions on Knowledge and Data Engineering **27**(8), 2203–2216 (aug 2015). https://doi.org/10.1109/TKDE.2015.2405509

[16] Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering expressive process models by clustering log traces. IEEE TKDE (2006)

[17] Heckerman, D.: A bayesian approach to learning causal networks. arXiv preprint arXiv:1302.4958 (2013)

[18] Leemans, M., van der Aalst, W.M.: Discovery of frequent episodes in event logs. In: Lecture Notes in Business Information Processing. vol. 237, pp. 1–31. Springer, Cham (11 2015)

[19] Leemans, S.J., Fahland, D., van der Aalst, W.M.: Discovering block-structured process models from event logs-a constructive approach. In: International conference on applications and theory of Petri nets and concurrency. pp. 311–329. Springer (2013)

[20] Leno, V., Dumas, M., Maggi, F.M., La Rosa, M., Polyvyanyy, A.: Automated discovery of declarative process models with correlated data conditions. Information Systems **89**, 101482 (mar 2020). https://doi.org/10.1016/j.is.2019.101482

[21] Li, J., Le, T.D., Liu, L., Liu, J., Jin, Z., Sun, B.: Mining causal association rules. In: Proceedings - IEEE 13th International Conference on Data Mining Workshops, ICDMW 2013. pp. 114–123. IEEE (dec 2013). https://doi.org/10.1109/ICDMW.2013.88, http://ieeexplore.ieee.org/document/6753910/

[22] Lorenzoli, D., Mariani, L., Pezzè, M.: Automatic generation of software behavioral models. In: Proceedings - International Conference on Software Engineering. pp. 501–510. ACM Press, New York, New York, USA (2008). https://doi.org/10.1145/1368088.1368157, http://portal.acm.org/citation.cfm?doid=1368088.1368157

[23] Maggi, F.M., Mooij, A.J., Van Der Aalst, W.M.: User-guided discovery of declarative process models. In: CIDM 2011. pp. 192–199. IEEE (4 2011)

[24] Maggi, F.M., Dumas, M., García-Bañuelos, L., Montali, M.: Discovering data-aware declarative process models from event logs. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 8094 LNCS, pp. 81–96. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40176-3_8

[25] Mendling, J., Reijers, H.A., van der Aalst, W.M.: Seven process modeling guidelines (7pmg). Information and Software Technology **52**(2), 127–136 (2010)

[26] Mendling, J., Reijers, H.A., Cardoso, J.: What makes process models understandable? In: International Conference on Business Process Management. pp. 48–63. Springer (2007)

[27] Nadkarni, S., Shenoy, P.P.: A bayesian network approach to making inferences in causal maps. European Journal of Operational Research **128**(3), 479 – 498 (2001). https://doi.org/https://doi.org/10.1016/S0377-2217(99)00368-9, http://www.sciencedirect.com/science/article/pii/S0377221799003689

[28] Neuberg, L.G.: CAUSALITY: MODELS, REASONING, AND INFERENCE, by Judea Pearl, Cambridge University Press, 2000. Econometric Theory **19**(04), 675–685 (aug 2003). https://doi.org/10.1017/s0266466603004109

[29] Pearl, J., et al.: Models, reasoning and inference. Cambridge, UK: CambridgeUniversityPress (2000)

[30] Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Transactions on Knowledge and Data Engineering (2004)

[31] Rabatel, J., Bringay, S., Poncelet, P.: Mining sequential patterns: a context-aware approach. In: Advances in Knowledge Discovery and Management, pp. 23–41. Springer (2013)

[32] Rozinat, A., Van Der Aalst, W.M.: Decision mining in ProM. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 4102 LNCS, pp. 420–425. Springer Verlag (2006). https://doi.org/10.1007/11841760_33

[33] Schönig, S., Di Ciccio, C., Maggi, F.M., Mendling, J.: Discovery of multi-perspective declarative process models. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 9936 LNCS, pp. 87–103. Springer Verlag (oct 2016). https://doi.org/10.1007/978-3-319-46295-0_6

[34] Senderovich, A., Weidlich, M., Gal, A.: Temporal Network Representation of Event Logs for Improved Performance Modelling in Business Processes. BPM **1**, 3–21 (2017)

[35] Shraga, R., Gal, A., Schumacher, D., Senderovich, A., Weidlich, M.: Inductive context-aware process discovery. In: Proceedings - 2019 International Conference on Process Mining, ICPM 2019. pp. 33–40. Institute of Electrical and Electronics Engineers Inc. (jun 2019). https://doi.org/10.1109/ICPM.2019.00016

[36] Silverstein, C., Brin, S., Motwani, R., Ullman, J.: Scalable techniques for mining causal structures. Data Mining and Knowledge Discovery **4**(2-3), 163–192 (2000). https://doi.org/10.1023/A:1009891813863

[37] Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Applications and Theory of Petri Nets. vol. 6128 LNCS, pp. 226–245. Springer, Berlin, Heidelberg (2010)

[38] Song, M., Günther, C.W., Van Der Aalst, W.M.: Trace clustering in process mining. In: Lecture Notes in Business Information Processing (2009)

[39] Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: International Conference on Extending Database Technology. pp. 1–17. Springer (1996)

[40] Sun, Y., Bauer, B., Weidlich, M.: Compound trace clustering to generate accurate and simple sub-process models. In: Maximilien, E.M., Vallecillo, A., Wang, J., Oriol, M. (eds.) Service-Oriented Computing - 15th International Conference, ICSOC 2017, Malaga, Spain, November 13-16, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10601, pp. 175–190. Springer (2017). https://doi.org/10.1007/978-3-319-69035-3_12, https://doi.org/10.1007/978-3-319-69035-3_12

[41] Tax, N., Dalmas, B., Sidorova, N., van der Aalst, W.M., Norre, S.: Interest-driven discovery of local process models. Information Systems **77**, 105–117 (9 2018)

[42] Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.: Mining local process models. Journal of Innovation in Digital Ecosystems **3**(2), 183–196 (2016)

[43] Thomsen, E.: OLAP solutions: building multidimensional information systems. John Wiley & Sons (2002)

[44] Van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H., Weijters, A., van Der Aalst, W.M.: The prom framework: A new era in process mining tool support. In: International conference on application and theory of petri nets. pp. 444–454. Springer (2005)

[45] van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Avoiding Over-Fitting in ILP-Based Process Discovery. In: BPM, pp. 163–171. Springer, Cham (2015)

Mehdi Acheli holds a degree in computer science specializing in computer systems. He is currently a Ph.D. student at Université Paris-Dauphine-PSL in Paris, France. Within the pole data science, he works on data mining, data analytics and process mining with a particular emphasis on flexible processes.

Daniela Grigori is a Professor of computer science at University Paris-Dauphine-PSL since September 2011 and she is currently the head of LAMSADE laboratory. Her current research interests include process management, data and process analytics, data integration. She has a number of papers in international conferences and journals and has served as organizer and program committee member in many conferences. She is the co-author of a book on process analytics.

Matthias Weidlich is professor and Chair of Databases and Information Systems at the Department of Computer Science at Humboldt-Universität zu Berlin. His research focuses on process-oriented and event-based information systems, and his results appear regularly in premier conferences (SIGMOD, VLDB, ICDE, IJCAI, BPM) and journals (TKDE, Inf. Sys., VLDB Journal) in the field. He is a Junior-Fellow of the German Informatics Society and in 2016 received the Berlin Young Researcher Award. He serves as editor-in-chief for the Information Systems journal and is a member of the steering committee of the ACM DEBS conference series.