Software and Systems Modeling (2023) 22:1755–1777
https://doi.org/10.1007/s10270-023-01100-w

SPECIAL SECTION PAPER

Detecting cross-case associations in an event log: toward a pattern-based detection

Yael Dubinsky[1] · Pnina Soffer[1] · Irit Hadar[1]

Received: 28 February 2022 / Revised: 11 October 2022 / Accepted: 13 February 2023 / Published online: 1 April 2023

Abstract
Business process management, design, and analysis is mostly centered around a process model, which depicts the behavior of a process case (instance). As a result, behavior that associates several cases together has received less attention. Yet, it is important to understand and track associations among cases, as they bear substantial consequences for compliance with regulations, root cause analysis of performance issues, exception handling, and prediction. This paper presents a framework of cross-case association patterns, categorized as intended association patterns and contextual association patterns. It further conceptualizes two example patterns—one for each category, and proposes techniques for detecting these patterns in an event log. The "split-case" workaround is an example of a pattern in the intended association category, and its proposed detection method exemplifies how patterns in this category can be approached. The patterns of a shared entity and a shared resource are contextual association patterns, which we propose to detect by means of hidden concept drifts. Evaluation of the two detection approaches is reported, using simulated logs for assessing their internal validity as well as real-life ones for exploring their external validity.

Keywords Process mining · Cross-case patterns · Split-case workaround

1 Introduction

Business process management (BPM) has become an operational backbone in modern organizations. As such it has received much attention from industry and academia. Along the years the focus of the BPM discipline has been centered around a process instance (termed a process case). The main BPM artifact, a process model, specifies the expected behavior of a process case, and serves as a basis for analysis, design, execution, monitoring, and evolution of a business process. Process mining, which is engaged with various process analysis tasks based on event logs, is also dominated by case-centric approaches. According to the well-accepted XES standard for event logs [1], an event log is comprised of traces of events, where each trace holds a sequence of events that describe the execution of one specific instance, or case, of the logged process. Accordingly, the main tasks accomplished by process mining approaches that build on XES logs are case-centric. Process discovery deals with the automatic creation of a process model, which captures case behavior based solely on the information available in an event log [2]. Conformance checking deals with measuring the extent to which actual case behavior, as recorded in the log, conforms with the expected case behavior specified in a process model (which can be given or discovered to capture mainstream behavior), identifying and diagnosing conformance violations. Specific process-mining-based analyses, such as performance assessment and prediction, are all case-centric, referring to and analyzing process case behavior.

While relying on information that is aggregated from all cases in a log, the focus of most process mining approaches is on within-case behavior, consequently, the vast majority of process mining approaches are not geared towards analyzing behavior that ties a number of cases together and associates them to one another. Examples of such behaviors include bundling a set of cases in a batch, employing decision rules based on some aggregated measure of recent cases (e.g., the total amount granted since the beginning of the month), and choosing among cases that compete for a position, a resource,

Communicated by Selmin Nurcan, Rainer Schmidt, and Adriano Augusto.

✉ Pnina Soffer
spnina@is.haifa.ac.il

[1] Department of Information Systems, University of Hafa, Haifa, Israel

or an award. Detecting cross-case behaviors is of importance for several purposes. First, the need for process discovery is not limited to within-case behavior, as many different practices involve more than one case. Second, organizational policies and rules may relate to association among process cases. To check the compliance of the actual processes with such rules, cross-case associations need to be detected and analyzed. Third, detecting and analyzing delays and problems that are related to cross-case behavior is important for improving the process. Last, prediction of future properties of a running case (e.g., time to completion, success or failure) depends on the environment of this case, including other cases that run around it. Understanding associations among different cases should contribute to the accuracy of prediction techniques, e.g., by predicting delays due to the load of shared resources.

Despite the relevance of these purposes, very few works have tackled cross-case behavior in the context of process mining so far. Examples include identification of specific cross case constraints [3], identification of batching behavior, where a number of cases synchronize for an activity which handles them together [4], and consideration of inter-case properties for prediction of future behavior of a process case [14].

In a paper presented in BPMDS 2021 [6] we proposed a method for detecting a specific pattern of cross-case behavior—the "split-case" workaround—in an event log. This workaround is common in processes where strict regulations require additional actions (typically inspection and approval actions) in cases whose volume (in terms of quantity or financial value) exceeds a given threshold. To avoid these additional actions, employees "split" the case into several cases whose volume is below the threshold, and these are the cases that then appear in the event log. Detecting this behavioral pattern in a log is not possible when each trace is analyzed separately. Rather, detection can only be accomplished by establishing associations among cases [6]. In the current paper we extend the BPMDS paper as follows. First, we present a framework of cross-case association patterns, categorized as intended association patterns, where association stems from intended actions and decisions of process participants, and contextual association patterns, where the association stems from a shared context. We discuss the differences between these two categories and the implications of these differences on how they can be detected in an event log. We use the "split-case" workaround as an example of a pattern in the intended association category, whose proposed detection method exemplifies how patterns in this category can be approached based on a clear conceptualization of the pattern. Considering contextual association patterns, we argue that a generic detection approach is needed, as a detailed pattern conceptualization may not be possible. We propose such a generic detection approach and evaluate the proposed approaches experimentally.

The remainder of this paper is organized as follows. Section 2 presents the framework of cross-case association patterns, which was developed based on a set of examples of instance-spanning constraints [7]. Section 3 discusses the "split-case" workaround as an example of the intended association patterns, and presents an algorithm for detecting this pattern in an event log. Section 4 turns to contextual association patterns and proposes an approach for detecting such behavior in an event log. Section 5 reports a set of experiments for evaluating the techniques presented in Sects. 3 and 4. Section 6 reviews related work and positions the current paper's contribution with respect to state of the art. Concluding discussion and future research directions are given in Sect. 7.

2 Cross-case association patterns

To characterize cross-case association patterns we relied on 114 examples of process behavior which may reflect interaction between cases of the same process or between interconnected processes [7], based on an extensive literature review. We focused on examples of behavior that associates different cases of the same process, leaving process interrelations out of scope. We also excluded associations that are manifested through aggregated KPIs (e.g., a KPI that measures the amount of cases executed per time unit), as these may serve for a post-hoc assessment rather than for steering process behavior at runtime. We have analyzed the remaining 53 examples, as well as additional examples drawn from literature and from our experience, as follows.

A first step included analyzing the 53 examples and coding them into an emerging set of patterns. For each example, a high-level generic description was formulated and the set of cases associated in the example was specified. When the high-level description of an example matched the description of previously analyzed examples, the examples were coded as pertaining to the same pattern. If an example could not be coded by an existing pattern, it would stand for a new one. This process was performed by one of the researchers, yielding 8 high-level patterns of cross-case association. A sample of 20% of the examples (11 examples) was used for validating the coding by a second researcher. The analysis and coding are illustrated by a sample of 5 examples given in Table 1, where the example numbers refer to their numbering in the original list [7].

Based on 15 additional examples drawn from the researchers' knowledge as well as relevant literature (e.g., [8]), one additional pattern was formulated, resulting in a total of nine patterns, to which titles were assigned. Next, the patterns were categorized into two categories: patterns where

Table 1 Sample analysis of cross-case behavior examples

No	Example	High-level description	Applies to	Comments
6	A user is not allowed to do t2 if the total loan amount per day exceeds 1 M$	A decision rule based on a case-aggregating function	Cases executed on the same day	Aggregating function: total loan amount
99	An employee is allowed to give a customer up to 10% discount. The monthly average discount shall not be over 10%	A decision rule based on a case-aggregating function	Cases executed by the same employee on the same month	Same pattern as (6). Aggregating function: monthly avg. discount
22	Prioritization and dynamic handling of cargo-vehicle connection to ensure precedence of high-priority cargo items over low-priority items	Prioritization policy of cases that require a shared resource	Cases that require the resource	
95	Upon full payment all instances of the file will be closed immediately	Cases share a goal. Once it is achieved by one, other cases are closed	Cases that refer to the same payment	Shared goal: payment
102	There was a problem with a product and now all instances are recalled	An exception in one case is propagated to others	Cases of the same product	

association among cases is intentionally created by process participants (intended association) and patterns where the association is a result of the cases' context, within the process or in its environment (contextual association). The distinctive criterion can be described as follows: when a pattern involves an intentional decision made by a process participant, considering a set of cases and binding them together, the pattern is classified as an intended association. Otherwise, the association is considered contextual. We note that this distinction can be delicate. For example, when a set of cases shares a resource or a set of resources, we consider this an association related to their context, unless an intentional decision is involved in prioritizing the cases or allocating the shared resources among them (e.g., example 22 in Table 1). If such a decision is made, the association is intentional. The patterns are presented in Table 2 (intended association) and Table 3 (contextual association).

As our aim is to detect behavior that involves cross-case association in event logs, we note that the challenges posed by intended association patterns are substantially different than those related to contextual association patterns. Intended association patterns often relate to a limited set of cases, and can be conceptualized and formalized, as the behavior in these cases is well defined. Accordingly, specific techniques can be developed for detecting each specific pattern. Indeed, designated techniques have been proposed for detecting batching behavior [4], aggregated decision rules [9], and specific resource allocation rules [10]. In all these cases, the techniques specifically address some well-defined behavior and are tailored based on a conceptual-level understanding of this behavior. In contrast, for contextual association patterns, the behavior that stems from the association is not necessarily known and well-defined. Some conceptualization can be made, e.g., for association due to a shared resource of a limited capacity. However, this will not cover resources whose capacity does not limit the process, that may still pose some form of association among process cases. It follows that an approach that would detect contextual association among cases in an event log needs to be generic and not tailored towards a specific behavior.

In this paper we present detection approaches of the two kinds. As an intended association pattern, we focus on the "split-case" workaround, show a detailed conceptualization of this behavior, and propose an algorithm tailored for detecting this pattern in an event log (Sect. 3). We selected this pattern as our focus for the following reasons. First, although this is a well-known behavior which bears significance in business terms, we are not familiar with any automated detection technique of this behavior in a log. Second, unlike other intended association patterns, it does not stand for an intended constraint or normal behavior that is expected to be observed in the log. Rather, it stands for a workaround, which, while not forbidden altogether, is not expected to be the normative behavior. Hence, its detection is challenging as compared to other intended association patterns. For the contextual association patterns, we propose in Sect. 4 a generic approach, capable of detecting association due to a shared entity or a shared resource, without a clear conceptualization of how this association manifests itself in the process behavior.

Table 2 Cross-case behavioral patterns: intended association

Pattern	Description	Set of associated cases	Example
Aggregated decision rule	A decision rule which applies to an aggregation of a data value over a defined set of cases	Denoted as part of the rule	The aggregated loan amount granted to a customer should not exceed 100 K$ a month
The "split case" workaround	Process participants create a number of (small) cases instead of one (large case) to avoid unwanted paths	The cases that are created by splitting an original case	A purchase request for 10,000$ is split into two requests of 5000$, avoiding the need for additional approvals for requests over 7000$
Batching behavior	A specific batching activity where cases synchronize so the activity is performed to all cases together	Cases in the batch (the same activity at time proximity)	The shipment of goods is optimized by batching goods to be transported to the same place
Step redundancy	When cases apply to the same specific object, duplicate steps can be skipped	Cases applying to the same object at time proximity	Several checks for a car are running at the same time. Similar tasks in different checks should be only executed once
Resource allocation rule	A rule which constrains resource allocation among cases (including case prioritization)	Denoted as part of the rule	There should not exist more than 4 cases where the customer is the same and the user executing t2 is also the same
Shared goal	Parallel cases intended to achieve the same goal. Once the goal is achieved by one case, others can stop	Cases aiming at the same goal, time proximity	Several diagnoses for a dysfunctional machine are running. If the problem is identified in one diagnosis, the others are cancelled

Table 3 Cross-case behavioral patterns: contextual association

Pattern	Description	Set of associated cases	Example
Association due to a shared resource	Many cases share a resource, and can be affected by its state or availability	Cases that share a resource	All the cases handled by Sarah cannot progress when she is ill
Association due to a shared entity	Many cases refer to an entity (represented by an instance of a data object), and can be affected by changes in its state	Cases that share an entity (data object instance)	Upon bankruptcy of a customer, all associated unpaid orders are considered lost debts
Contamination	An exception in one case may "spread" and affect nearby cases	Based on time and space proximity	During a production process a blade got lost. Now all the produced items will be searched until the blade is found

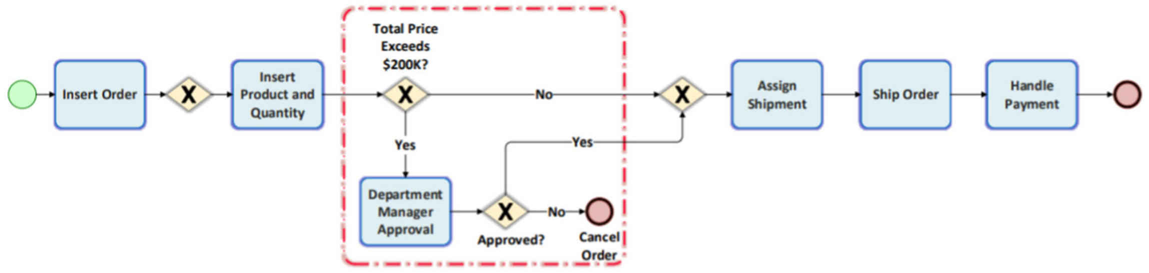


Fig. 1 An order handling process where the split-case workaround may exist

3 Detecting the "split-case" workaround

The "split-case" workaround is a common workaround in business processes [11], which involves splitting one case into several ones, so each case separately is "eligible" to an easier, preferable path. As an example, consider the process depicted in Fig. 1, where orders whose total price is over 200 K$ need to be approved by a manager, and this may cause delays in the process. When time is of essence, an employee may decide to split an order whose price is over 200 K$ to two or more smaller orders, thus avoiding the need for approval. Detecting split cases in processes of this kind is essential, as the approvals and tasks that are avoided are likely to be crucial for valid process execution, for reducing business risks, and for complying with regulations and legislation. Yet, this kind of behavior cannot be detected by existing conformance checking techniques [12], as each one of the resulting ("split") cases is considered conformant by itself. This behavior can only be detected when analyzed as a cross-case pattern.

3.1 Pattern conceptualization

As the split cases behavior may occur only under certain circumstances, where a motivation for splitting cases exists, we start by formulating a set of assumptions that should hold in a process for such behavior to be plausible. The formulation relies on the Petri net with data (DPN) formalism, which has been defined as follows [13].

Definition 1 (Petri net with data). A Petri net with data $DPN = (P, T, F, D, U, R, W, G)$ consists of:

- a Petri net (P, T, F), where P is a set of places, T a set of transitions, and F a set of flows;
- a set D of data objects;
- a function U that defines the domain of values admissible for each data object $d \in D$;
- a read function $R \in T \rightarrow 2^D$ that labels each transition with the set of data objects it reads;
- a write function $W \in T \rightarrow 2^D$ that labels each transition with the set of data objects it writes;
- a guard function $G \in T \rightarrow \Phi(d)$ that associates a guard with each transition, where $\Phi(d)$ is a predicate in d.

For developing our approach, we now turn to the premises that should hold for creating a possible motivation to engage in case splitting:

(1) There exists a value function V which assigns a partial order relation to traces σ_i, σ_j, so it can be said that $V(\sigma_i) > V(\sigma_j)$. Note that this function is not necessarily explicit or even known, but it marks that an employee may favor one possible process path (and resulting trace) over another. Considerations may relate to time, effort, certainty of the outcomes, or even personal relations with other employees. In the order handling example (Fig. 1), the path without a manager's approval would be preferable over the path that includes it, for time and certainty considerations.

(2) There exists a data object d and at least two transitions t_1,t_2 (each marking or leading to a different process path), such that different guard conditions over d are set for t_1 andt_2. Formally:$G(t_1) = \Phi_1(d), G(t_2) = \Phi_2(d), \Phi_1(d) \neq \Phi_2(d)$. The relevant data object in the order handling example is the total price of the order.

(3) Consider two transitionst_1,t_2, that satisfy (2). For two traces σ_1, σ_2, such that$t_1 \in \sigma_1$,$t_2 \in \sigma_2$, and t_1, $t_2 \notin \sigma_1 \cap \sigma_2$, $V(\sigma_1) \neq V(\sigma_2)$ holds. We then say that V depends on$d$. In other words, the value of d may determine whether a preferable path is taken or not. In the order handling example the selection of a preferable path depends on the total price of the order.

The realization of the above premises in a process may motivate the creation of split cases. A typical scenario is of a case whose value of the decision data object d should lead to an undesirable path. To avoid this and improve the value function V, an employee may split the case by creating several cases instead, each having a d value that warrants a preferable path, where the total sum of d over the split cases would be equal to the original d. In the log generated by these process executions, the original case would not be present, and it is hence termed a *concealed case*. Instead, the log would include the *split cases* as formalized in Definition 2.

Definition 2 (Concealed Case and Split Cases). Given an event log and a process model, a concealed case CC is an assumed case whose trace does not appear in the log. Instead, the event log includes traces of at least two split cases SC_i. Let σ_{cc} be an imagined trace fully aligned with the process model based on the data values of CC. Let d be a data object on which V depends. We denote the value of d for case C as $C.d$, then $CC.d = \sum_j SC_j.d$ and $V(\sigma_{CC}) < V(\sigma_{SC_j})$ for all j.

We base the detection approach presented next of the notions of concealed and split cases.

3.2 Detection of split cases in an event log

Detecting split cases in an event log requires a preliminary identification of the elements that indicate that our three premises hold and the behavior can indeed be motivated in the investigated process. These include a data value d on which

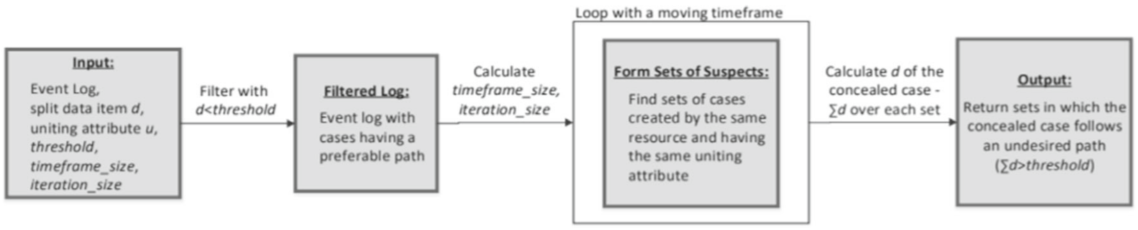

Loop with a moving timeframe
Input:
Event Log, split data item d, uniting attribute u, threshold, timeframe_size, iteration_size
Filter with d<threshold
Filtered Log:
Event log with cases having a preferable path
Calculate timeframe_size, iteration_size
Form Sets of Suspects:
Find sets of cases created by the same resource and having the same uniting attribute
Calculate d of the concealed case - ∑d over each set
Output:
Return sets in which the concealed case follows an undesired path (∑d>threshold)

Fig. 2 An overview of the detection algorithm

some plausible value function depends, and the threshold values of d for selecting the process path.

The value of d should be known upon case creation for the motivation to split cases to exist. We note that identification of data objects that affect decision rules can be done using existing techniques, e.g., as suggested by De Leoni and Aalst [13], or rely on domain knowledge. The threshold conditions over d may appear explicitly as part of guard conditions that can be extracted from a process model if such is available. Alternatively, they can be learned from log data using decision mining [13] or similar techniques. Furthermore, we expect the traces that follow values below and above the threshold to be differently ranked according to some value function V. Note that V is typically unknown when attempting to detect split cases, and domain knowledge might be needed for suggesting a plausible one (e.g., a smaller number of activities is more preferable than a larger one). Our assumption is that values above the identified threshold would be considered less desirable (so splitting the cases and reducing d in each case would lead to a preferable path).

Based on our conceptualization, the detection approach identifies sets of cases that are suspected as being split (from an original concealed case), using the following criteria:

- All cases follow a "preferred" path according to a perceived value function V.
- All cases are created by the same resource—the one who decided to split them.
- All cases are created in time proximity to one another. The rationale is that once a split decision is made, all the cases would be created shortly. To accommodate for this, the algorithm looks for cases within a defined time window, set as a parameter by the user.
- The sum of d over all the cases exceeds the threshold, namely, its original value in the concealed case would result in a path which is less desirable (based on V).
- All cases share values of at least one "uniting attribute" in addition to the shared resource. As the split cases hold the information of the concealed case , they are expected to have identical values for attributes that characterize that case (e.g., customer). Such attributes serve the detection method for uniting the split cases, and need to be identified beforehand.

We note that different scenarios may be expected for the uniting attributes. For example, a customer order can be split so each new order will have part of the quantities ordered in the concealed case, resulting in split cases that have the same products, the same due dates, and the same customer. Alternatively, different products and due dates may be assigned to different split cases, yet all cases share the same customer. It follows that we expect at least one shared attribute for the split cases, that would indicate an originating concealed case. This "uniting" attribute can be selected beforehand based on domain knowledge, but can also be inferred from the data. Note that the resource who creates the cases is shared among the split ones by our premise that it is this resource who initiates the splitting. The uniting attribute should thus be common to the split cases in addition to the resource.

The algorithm, depicted in Fig. 2 and listed as pseudocode in Listing 1, filters the log to include only the cases where the value of d is below the threshold. It then iterates over the log with a moving time frame, examining the cases within the timeframe according to the above specified criteria. It returns identified sets of cases that meet all the criteria and are hence suspected of being split (from some concealed case). The parameters of timeframe and iteration size (the amount of time by which the timeframe is moved in each iteration) are set by the user with respect to the Mean Time Between Cases (MTBC) as a factor, e.g., $Timeframe = F*MTBC$.

Listing 1.The split-case detection algorithm.

```
Input: event log E, data object d, condition threshold value
threshold, uniting data object u, timeframe size timeframe_size,
iteration size iteration_size.
Output: a set of sets S2, where each set contains cases suspected
as splitted.
Begin
1. E1 ← E filtered on cases where d < threshold
2. E2 ← E1 filtered on first event of each case
3. S ← {}
4. For frame_start = minimal timestamp in E2;
   frame_start + timeframe_size < maximal timestamp in E2;
   4.1.      E3' ← E2 filtered on timestamp between frame_start
      and frame_start + timeframe_size
   4.2.      E3'' ← E3' sorted by resource
   4.3.      For each resource r in E3'':
      4.3.1. For each value of u, $u_i$:
         4.3.1.1.  RS ← {}
         4.3.1.2.  RS ←{g | g contains cases from E3'', where
                resource=r
                and u= $u_i$}
# set of cases created in time proximity from one another, by the
same resource, and hold the same uniting attribute.
         4.3.1.3.  S ← S + RS
   4.4.      frame_start = frame_start + iteration_size:

5. For each set g in S:
   5.1.      aggregate_split_col ← $\sum_{case\ in\ g} d$
   5.2.      If aggregate_split_col > threshold:
      5.2.1. S2 ← S2 + g
6. Return S2
End
```

4 Detecting contextual association among cases

While patterns of intended association of cases can be formally conceptualized, as exemplified in the previous section, contextual association may be manifested in many different ways, not always explicitly visible. Consider, for example, a set of cases where a shared external entity (e.g., a customer, a supplier, a raw material) is involved. Assume a customer is implementing new quality policies—then all the cases related to her orders will have to follow stricter quality requirements, perhaps involving additional quality checks. If, alternatively, the customer faces financial difficulties, then payment in advance will be required for the relevant cases. Cases that share some raw material whose availability is limited might be delayed while waiting for the material to arrive, might use some replacement material and suffer quality issues, or may need to be prioritized one over the other. In summary, the manifestation of association among cases due to some shared contextual object may vary. Yet, we observe in all these examples that when the state of the shared contextual object changes, all the related cases may be affected. Such effects may apply to each case separately, resulting in correlation in their behavior, and may trigger interaction among the cases due to the shared object.

As the kind of interaction that may be triggered is not always expected, we rely on the behavioral correlation to identify contextual association. Our premise is that contextual association among cases would be manifested as *concept drifts* in subsets of associated cases, reflecting some contextual change in the state of an object that is related to each subset, while the actual change is not necessarily known. Identifying such concept drifts would indirectly indicate the existence of a contextual association among the related cases.

4.1 Drifts as a manifestation of contextual association

As our assumption is that contextual association among cases may exist, induced by contextual objects that are shared by the associated cases, the question we address is what are the objects that induce contextual association. More specifically, concerning specific contextual objects we ask whether they induce contextual association among cases. To establish our detection approach, we make the following definitions:

Definition 3 (Shared object). An object *O* that has a one-to-many or a many-to-many relation with the case object of the process is a shared object.

For example, in an order fulfillment process, the Customer object has a one-to-many relation with the Order object—which is the case object of this process. From Definition 3 it follows that each instance of a shared object (e.g., each specific customer) is referred to by a set of cases—the many cases where it participates.

Definition 4 (Sharing set). A set of cases where the same instance of *O* participates is a sharing set of *O*.

In the order fulfillment example, all the cases where orders of John Smith are handled are in one sharing set of Customer. If *O* induces a contextual association, this will apply to cases in sharing sets of *O*. This may be observable in the form of a *hidden drift*.

Definition 5 (Hidden drift). . A drift that locally affects a sharing set of an object *O*, while not affecting other cases of the process, is a hidden drift with respect to *O*.

As a result, a hidden drift is not apparent while examining the whole log of the process, yet it is apparent while examining the log records associated with a sharing set of *O* separately.

We focus on two kinds of drifts that can result from a change in the state of a contextual object: a concept drift—a change in the control flow of the process, and a time drift—a change in the time of a certain activity, process part, or the entire process. For both we assume the drift only holds for a subset of cases—a sharing set of *O*—while the process in general has no observable drift.

4.2 Detection approach

The proposed detection approach attempts to discover a hidden drift, either concerning the control flow of the process or the time dimension. For simplicity, and for avoiding long tail effects, time drift focuses on the median time duration of activities. As hidden drifts affect a subset of the cases, they are best recognized in logs that do not have drifts for the full set of cases. The approach considers a given object as the contextual object whose possible role as inducer of association among cases is assessed. The object to be considered may be selected based on domain knowledge, or it is possible to apply the approach to different objects, which exist in the log and have a one-to-many or many-to-many relation with the case object, one after the other. If hidden drifts are detected for a set of cases that share this object, it can be concluded that this object induces a contextual association among cases.

For the investigated object, the proposed algorithm (see Listing 2) partitions the log into a set of logs, each holding cases of a sharing set of that object (corresponding to a unique instance of the object). For example, if the contextual object is Customer, then each sharing set (and each of the resulting logs accordingly) holds cases that involve a single customer. To avoid the creation of small logs that hold just a few cases each, we use a MinCases parameter which denotes the minimal number of cases in a log for it to be investigated. Then, for each of the resulting logs, the algorithm calls two main functions—one is a concept drift detection algorithm, and the other attempts to detect time drifts in the log. The algorithm returns two sets of couples, C for identified control flow concept drifts and T for identified time drifts. Each couple is of the form (v,S), where v is a value of the object O and S is a set of indices where drifts are identified.

For the concept drift detection function called by the algorithm we use an existing concept drift detection algorithm presented by Martjoshev et al. [14], which is available as a PROM plugin. It was selected as it is capable of identifying concept drifts and locating them with high precision for varied log sizes. The time drift detection function is depicted in Listing 3. It receives an event log filtered for a specific value of O, filters it on each activity and partitions the filtered logs to sublogs of a size given as a parameter (sublog size). The median durations of the investigated activity in each sublog form a sequence of values, where exceptional deviations (over two standard deviations [4]) between following sublogs are considered as indicating a time drift.

Listing 2.Hidden drift detection algorithm with respect to object O.

```
Input: event log E, object O, MinCases, concept drift parameters
{min, max, step, p-value}, time drift sub-log size
Output: C, T: sets of values that induce concept drift(C) and time
drift (T), where each value is coupled with a set of drift indices in
the log)
Begin
1. For each value v of O
   1.1.      E(v) ← E filtered on cases where for at least in one
      event O=v holds
   1.2.      If number of cases in E(v)< MinCases discard E(v)
2. For each E(v)
   2.1.      P(v) ← Drift points found by concept drift plugin with
      parameters v, min, max, step, p-value, executed against E(v)
   2.2.      If P(v)≠Ø C ← {C,(v,P(v))}
   2.3.      TD(v) ← TimeDrift(E(v), sub-log size)
   2.4.      If TD(v) T ← {T,(v,TD(v))}
3. Return C, T
```

Listing 3. Time drift detection algorithm.

```
Input: event log E, object O, object value v, sub-log size S.
Output: a boolean value true/false; If true, returns activities and
drift indexes.
Begin
1. RS ← {}
2. For each activity a:
   2.1.   E* ← E filtered on activity=a
   2.2.   $E^*_1,...,E^*_n$ ← sub-logs of E* of size S
   2.3.   $D^*_1,...,D^*_n$ ← median durations of sub-logs $E^*_1...E^*_n$
   2.4.   SD ← Standard deviation($D^*_1,...,D^*_n$)
   2.5.   Plt* ← plot with x-axis={1,...,n}, y-axis={$D^*_1$, $D^*_n$}
          # set maximum value of y-axis to the average duration found
          in E*, or the highest median, the greater of them
   2.6.   For i=2...n if $|D^*_i - D^*_{i-1}| > 2*SD$:
          T* ← {T, i}
   2.7.   If T*≠Ø:
          RS ← {RS,(a, T*)}
3. If RS≠Ø return true, return RS
   Else, return false
End
```

5 Evaluation

To evaluate the approaches proposed in the paper, we conducted a series of experiments, separately addressing the detection of split cases and the detection of contextual association via hidden drifts. Each detection approach was evaluated in a controlled setting, using simulated data, as well as by applying it to real-life logs available publicly. In the experiments that use simulated data the ground truth is known, and the detection results can be assessed in terms of precision, recall, and F1. However, this data may not encompass the complexity and magnitude of real-life data. The studies that use the publicly available real-life logs allow us to evaluate the applicability of the approach in realistic setting and its ability to provide results, albeit in the absence of a ground truth, further assessment is not possible. In what follows we report the experiments performed for each of the detection approaches.

5.1 Evaluation of the split-case workaround detection approach

5.1.1 Experiments using simulated Logs

Aim: The aim of these experiments was threefold. First, as these are controlled experiments with a known (simulated) ground truth, we wanted to test the performance of the approach in terms of precision, recall, and$F1$. Second, as the algorithm uses input parameters of timeframe size and iteration size, we wanted to assess the sensitivity of the detection to the values of these parameters. Furthermore, we expect that as the timeframe size is increased, the recall would increase until some stability would be reached; at the same time, precision may start to decrease for larger timeframe sizes. We hence expect the F1 measure to indicate an optimal value of this parameter. Last, as the algorithm involves iterations over the log with a moving (relatively short) timeframe, scalability to large logs might be an issue and result in long execution times. To test this, we measured the execution times for the different configurations of timeframe and iteration sizes. To accomplish these goals, we conducted two sets of experiments, each focusing on a different aspect of the parameters setting.

Setting: We created simulated logs that follow the order handling process used as a motivating example in Sect. 3 (Fig. 1). Nine simulated event logs were created for this experiment, each simulating a period of one year. Seven logs represent normal process executions with a steady rate of case arrival, but with different frequencies of split cases. The last two logs represent specific scenarios of process executions, which may be difficult for the detection method to handle and yield false positives. One specific scenario assumes account executives: employees who dedicatedly handle the same customers. Considering the customer ID as a uniting attribute, this might mislead the detection method to identify sets of cases that have resource–customer matchings as split cases. The second specific scenario relates to busy work days, when the number of created cases reaches a peak. In an irregularly busy work day a large number of cases are created within a short time. The simulated log related to this behavior included 15 busy days along the year, while the rest 350 days behaved normally. The time between creations of split cases in all simulated logs ranges from 1 to 20 min, distributed uniformly. Table 4 summarizes the statistics of the simulated logs.

For the first set of experiments we kept a constant ratio between the timeframe size and the iteration size, and tested different values of the timeframe size. These parameters were calculated with respect to the mean time between cases (MTBC), as follows:

Timeframe size = X*MTBC; Iteration size = Y*MTBC (with $Y = X/4$). The algorithm was executed with different values of$X$ for all nine simulated logs. Our implementation was in Python via PyCharm IDE, and the experiments were conducted using a CPU of 12 threads, 6 cores, clock speed of 4.3 GHz, and a 16 GB RAM.

The second set of experiments focused on the iteration size. Using a fixed timeframe size, selected as the optimal$X$ value found in the first set of experiments, we tested the algorithm using different iteration sizes.

Findings: The precision, recall,$F1$, and average execution times that were obtained for different values of$X$ are shown in Table 5. The table shows the average precision and recall values for the seven logs that simulate normal behavior, and separately the values for the logs that simulate the specific scenarios.$F1$ values and execution times are presented as average for all logs. Note that our experiments did not yield any difference in the results related to different numbers of split cases in the log (log 6 and 7), hence we do not show these separately, but as part of the average results for the "normal behavior" logs.

As seen in Table 5, the recall values are similar for the "normal" behavior logs and the ones that simulate specific "difficult" scenarios. This result is expected, as the irregular behaviors (peak days and customer executives) may add false positives, but these do not affect recall values. Recall is low for low values of X and reaches values over 0.99 for $X \geq 0.4$, indicating that for this range of timeframe sizes the detection method is able to detect nearly all the split cases. As X stands for the ratio between the timeframe size and MTBC, small values of X imply that the timeframe size is too small for capturing split cases that were not created immediately one by the other. The MTBC in the simulated logs is approximately 50 min. Values of 0.1–0.3 of X imply timeframe sizes of 5 to 15 min, which means that only split cases that were created up to 5–15 min from one another are found by the algorithm. As split cases in the simulated logs are created up to 20 min from one another, it is reasonable that many of them were not detected with smaller timeframes.

The average values of precision for normal behavior logs are over 0.8 for all values of X, and reach a stable value of 0.946 for $X \geq 0.4$, implying that for this range of timeframe sizes the detection algorithm performs well in terms of precision as well and does not yield large numbers of false positive split cases. This result is understandable, as the iteration size increased proportionally to the timeframe size. With a fixed iteration size, the predicted results could have grown, and precision would have decreased.

Table 4 Simulated log used for split-cases experimentations

Event log file	Behavior	# of cases	# of events	# of split cases	Mean time between cases (min)
1	Normal	10,516	84,112	500	52:13
2	Normal	10,505	82,039	500	50:49
3	Normal	10,512	82,261	500	48:44
4	Normal	10,361	82,593	500	51:37
5	Normal	10,454	81,788	500	50:16
6	Normal	10,384	82,137	500	51:38
7	Normal	10,412	83,411	100	52:21
8	Account executives	10,497	82,645	500	50:08
9	Peak days	10,422	84,008	500	49:35

Table 5 Precision, recall,$F1$, and execution times for different timeframe sizes with a constant ratio between timeframe size and iteration size

X	Y	Precision			Recall			Avg$F1$	Avg execution time (min)
		Normal	Accounts executives	Peak day	Normal	Account executives	Peak day		
0.1	0.025	0.809	0.732	0.734	0.257	0.274	0.267	0.393	6:07
0.2	0.05	0.903	0.854	0.846	0.555	0.568	0.563	0.684	3:55
0.3	0.075	0.936	0.912	0.914	0.854	0.859	0.863	0.890	3:05
0.4	0.1	0.946	0.919	0.916	0.997	0.995	0.996	0.963	2:41
0.5	0.125	0.946	0.921	0.917	0.997	0.997	0.999	0.964	2:26
0.6	0.15	0.946	0.917	0.908	0.997	0.998	1.000	0.962	2:16
0.7	0.175	0.946	0.912	0.904	0.997	0.998	0.998	0.961	2:09
0.8	0.2	0.946	0.907	0.904	0.995	0.998	0.998	0.960	2:03
0.9	0.225	0.946	0.906	0.903	0.995	0.998	0.997	0.959	2:00

However, for the logs that simulate specific irregular scenarios, lower precision values were achieved, and a different behavior of the precision measure was observed. As expected, precision values increased with the increase of the timeframe size until a maximum point (achieved for $X = 0.5$), after which increasing the timeframe size resulted in decreasing precision values. This implies that higher timeframe sizes resulted in false positive identification of split cases. We still note that even the precision values obtained for $X = 0.9$ (the largest in the experiment) were still over 0.9.

The average $F1$ values, which combine precision and recall, increase with the timeframe size until a maximum value of 0.964 is reached for $X = 0.5$. Additional increases of the timeframe size lead to a (slight) decrease in $F1$, reflecting the decreasing precision. It can hence be concluded that $X = 0.5$ reflects an optimal value of the timeframe size in this setting, yet for every timeframe where $X \geq 0.4$ F is over 0.95.

The average execution times for this set of experiments are also presented in Table 5, showing that the execution time decreases as both X and Y increase. This is expected, as for higher values of X and Y less loops are executed. For the combinations of values that yielded$F1$ values of above 0.95 the average execution times were 2–2.5 min.

For the second set of experiments, we used a fixed timeframe ($X = 0.5$, following the first experiments' results) for testing different iteration sizes, calculated as Iteration size $= Y$*MTBC. Average precision, recall, and $F1$ values that were obtained, as well as the corresponding execution times are presented in Table 6.

In these experiments, expectedly the recall values for all logs decrease as the iteration size increases, as this means that the search along the log is less thorough. For precision, on the other hand, we observed different trends for the different logs. For the normally behaving logs, precision slightly decreased alongside the recall as the iteration size increased. For the two specific scenario logs, where the likelihood of false positive was higher as a starting point, precision increased with the increase of the iteration size. This can be explained by the reduction of false positive detection. Since the effect of

Table 6 Precision, recall, F1, and execution times for different iteration sizes with a fixed timeframe size

Y	Precision			Recall			Avg F1	Avg execution time (min)
	Normal	Accounts executives	Peak day	Normal	Account executives	Peak day		
0.1	0.948	0.907	0.903	0.997	0.995	0.994	0.956	03″34
0.3	0.948	0.914	0.916	0.934	0.941	0.938	0.932	01:12
0.4	0.944	0.922	0.923	0.812	0.821	0.816	0.869	00:54
0.5	0.935	0.928	0.031	0.674	0.669	0.780	0.780	00:44

changing the iteration size was stronger for recall than for precision, the $F1$ values decrease when the iteration size increases.

Similarly to the measurement of execution times for the first set of experiments, we measured execution times and checked their dependency on the iteration size. The results show that the execution time strongly decreases when the iteration size increases.

5.1.2 Empirical investigation using a real-life log

Setting: The suggested approach was further evaluated by application to a real-life log of a purchase order handling process in a large company operating from the Netherlands, in the area of coating and paints, which served for the 2019 BPI[1] challenge. We selected this log as it was possible to identify in this log a data item which could have impact on a plausible value function to motivate splitting cases, as explained below. The log contains four types of flows, each having different activities and business rules regarding the goods receipt messages and invoices. We focused on the “3-way matching, invoice after goods receipt” flow, which was filtered from the entire log by the Item Category attribute. The resulting log has 15,182 cases and 319,233 events, 38 activity types performed by 262 different users.

In order to identify the splitting decision rule (a threshold over a data object), we relied on the BPI challenge report [15], which indicated that cases whose Cumulative Net Worth is over 100,000€ take longer than cases below that sum. We filtered the log using Fluxicon DISCOvery[2] and compared the mean case durations for cases where Cumulative Net Worth was less than 100,000€ (mean case duration of 71.7 days) with those where Cumulative Net Worth was over 100,000€ (mean case duration of 100.1 days). As there was a substantial difference in the mean case durations, we suggest that this might be the basis of a value function, which motivates process participants to split purchase orders: an employee might prefer creating several ’fast’ purchase orders than one ’slow’ order.

The uniting attribute for the log was decided based on domain knowledge provided by the process owners, and selected to be the Vendor attribute. The input parameters of timeframe_size and iteration_size were set to 0.5*MTBC and 0.1*MTBC, respectively, as these were the optimal values found using the simulated logs. The MTBC in the event log was approximately 34 min.

Findings: Using the described inputs, 123 occurrences of the split case behavior were observed in the log, where each occurrence is a set of cases which are suspected as split cases. A total of 513 cases were marked by the method, out of 15,182 cases in the event log. The execution time for this log was 15:13 min. An example of a set of cases indicated by the method consists of three cases that were all created by user_091 in a period of 9 min, and are associated with vendorID_0183 (the last three rows in Table 7). On average, the Cumulative Net Worth of these cases is 37,000€, thus for the concealed case this would have indeed exceeded 100,000€, which is the assumed threshold.

To qualitatively assess whether the split cases pattern indeed occurs in this event log, we manually examined six cases created by user_091, associated with vendorID_0183 and the same item type, all created on January 2nd 2018, from 14:34 to 16:15. The six cases are presented in Table 7. We note that additional cases created by user_091 appear in the log, but not in time proximity to the ones in this set. This can indeed indicate that the detected cases are split. The three cases found by the method appear in the last three rows of Table 7. As the timeframe size we used was 17 min, these three cases were detected, while the others were not. Had we used a larger timeframe size, at least some of the other cases could be detected as well, but the risk of false positives would increase as well.

In summary, applying the method to this event log has shown that the suggested approach is capable of handling the scale and complexity of real-life event logs and detecting split-case behavior even when such is not known in advance to exist. As ground truth is not available for real-life logs, the detected sets of cases are merely suspected as split cases.

[1] https://doi.org/10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1.

[2] https://fluxicon.com/disco/.

Table 7 Cases suspected as split in a real-life event log

Case ID	Resource	Complete timestamp	Item	Purchasing document	Vendor	Cumulative net worth (EU)
4507000378_00010	user_091	1/2/2018 14:34	10	4,507,000,378	vendorID_0183	37,169
4507000417_00010	user_091	1/2/2018 15:51	10	4,507,000,417	vendorID_0183	35,832
4507000418_00010	user_091	1/2/2018 15:54	10	4,507,000,418	vendorID_0183	38,160
4507000421_00010	user_091	1/2/2018 16:06	10	4,507,000,421	vendorID_0183	33,058
4507000422_00010	user_091	1/2/2018 16:11	10	4,507,000,422	vendorID_0183	33,740
4507000425_00010	user_091	1/2/2018 16:15	10	4,507,000,425	vendorID_0183	48,054

When such would be detected in practice, additional investigations on site would be needed for validating this suspicion. Yet, the indicated cases form a starting point for such investigations.

5.2 Evaluation of the contextual association detection approach

5.2.1 Experiments using simulated logs

Aim: A main goal of the controlled evaluation based on simulated logs was to assess our capability to identify contextual association that may be induced by the same object but with different manifestations in the process behavior, as discussed in Sect. 2. To this end, we considered an order handling process (similar to the process presented in Fig. 1, excluding the approval path). Focusing on a product as a shared object, we examined two different scenarios. First, a scenario (Scenario 1) where a temporary shortage of a certain product leads the cases where this product is ordered to a *Wait for stock* activity after availability is checked, and this activity is not taken otherwise. This should be detected as a hidden concept drift for the relevant set of cases. Second, a scenario (Scenario 2) where shipments from a specific port are temporarily stopped, and this causes delays in the arrivals of a specific product. As a result, availability of the product cannot be immediately checked and the check might take longer than usual, as it may be necessary to wait for expected arrivals and to prioritize certain cases over others. This can be detected as one or two hidden time drifts for the *Check product availability* activity.

The controlled evaluation had two additional goals: one was to evaluate the time drift detection, which was developed as part of our algorithm, and the other was to assess the capability of the algorithm to identify contextual association in a log that includes more than one hidden drift, associated to more than one object instance.

Setting: For the controlled evaluation, we simulated three logs, whose details are summarized in Table 8. One log simulated Scenario 1 with one affected product (ProductID = 3). The log was simulated over a period of 1 year, containing normal process executions as well as a small amount of noise and irregularities, but no time drifts. The first case that reached the *Wait for stock* activity started on 2020/10/15 18:27, and the first occurrence of this activity was on 2020/11/04 18:30. The second log simulated Scenario 2 with one affected product (ProductID = 3). As a result of this scenario, the duration of the *Check product availability* activity for this product was simulated as taking 2–5 days, as compared to the normal duration of 5 h on average. The log was simulated over a period of 1 year, containing normal process executions as well as some noise and irregularities, and no control-flow concept drifts. The time drift was simulated during the month of March. The third log combined these two scenarios, with Scenario 1 simulated for ProductID = 3, and Scenario 2 simulated for ProductID = 2. The log was simulated over a period of 1 year, containing normal process executions as well as some noise and irregularities. Scenario 1 was simulated during the months October-December, and Scenario 2 was simulated for 1 month in June.

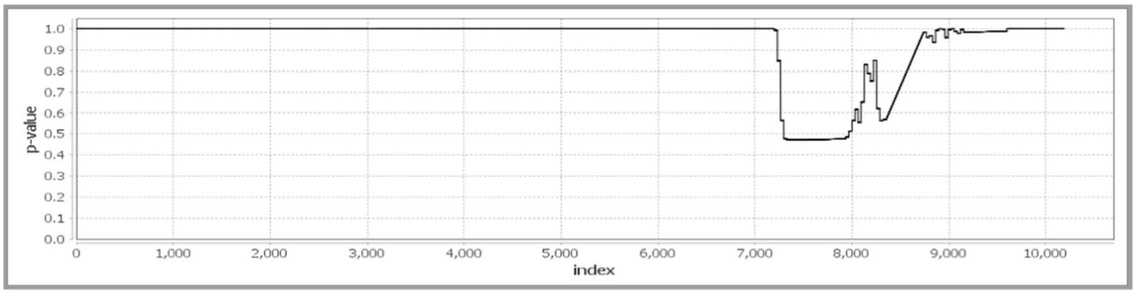
For setting the parameter values for the concept drift detection function, we followed the suggested ratios [16] with respect to the size of the log. Specifically, the ratio between the number of cases and the maximum window size was 12, the ratio between the maximum and the minimum window sizes was 6, and the step size was the minimum window size divided by 5. The p-value for the statistical tests performed by the concept drift detection function was 0.1. The sub-log size for the time drift function was the log size divided by 50, namely, approximately 200 for the full logs, and proportionally smaller for filtered ones.

We experimented with the detection algorithm using the three simulated logs. As a first step in each experiment, we verified that the full log did not exhibit a drift of any kind, so only hidden drifts could be detected. We then applied the algorithm with the product as the investigated object.

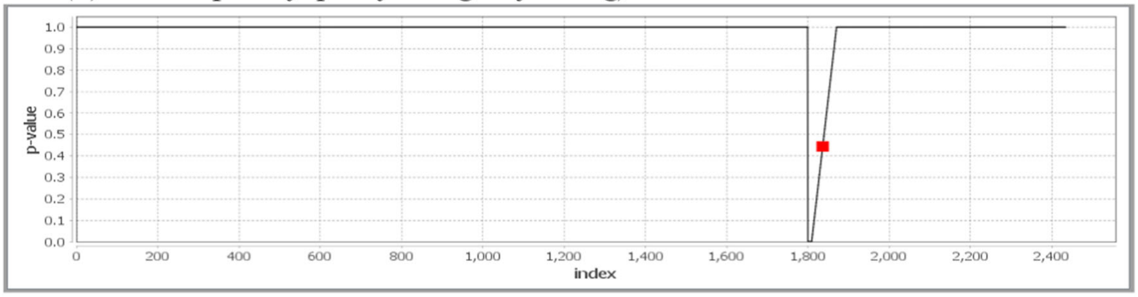
Findings: The results obtained for Log 1 show no indication of a time drift, and a positive indication of a hidden concept drift in the control flow for cases associated with Product 3. The concept drift plots for the entire log, the cases

Table 8 Simulated logs used for contextual association experiments

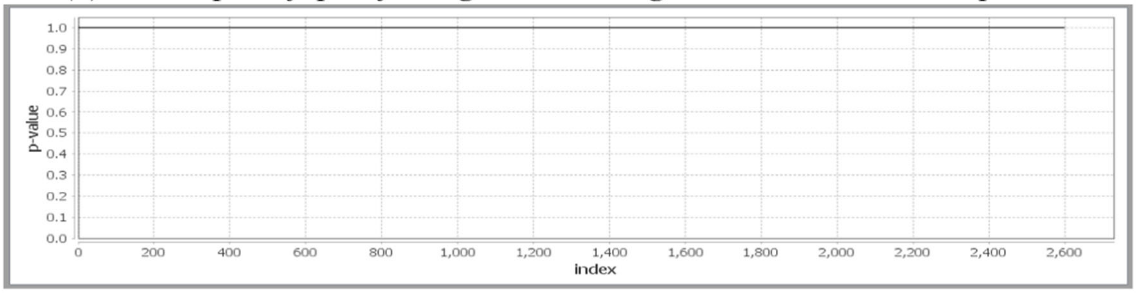
Log	Simulated behavior	Simulated period	Cases	Events	Activity types	Additional data
1	Scenario 1 (hidden control flow drift)	Year	10,188	62,838	11	6 users 4 products
2	Scenario 2 (hidden time drift)	Year	10,209	71,703	10	6 users 4 products
3	Scenario 1 and Scenario 2	Year	20,375	144,588	12	6 users 4 products



(a) Concept drift plot for log 1 (full log)



(b) Concept drift plot for log 1 concerning cases associated with product 3



(c) Concept drift plot for log 1 concerning cases associated with product 1

Fig. 3 Concept drift plots for log 1, p-values on Y-axis relate to a hypothesis that behavior has changed in that point. A red dot marks a positive indication of a concept drift

associated with product 3 and those associated with a different product (product 1) are given in Fig. 3.

As can be seen in Fig. 3a, while visually some change of behavior appears for the full log, it does not meet the p-value threshold and is not statistically significant. In contrast, the plot for cases associated with product 3 (whose concept drift was simulated) clearly indicates a significant concept drift on index = 1839 that corresponds to caseID = 8285, which started on 2020/10/14 10:27. This is in fact very close to the start time of the simulated scenario (the ground truth), which was 2020/10/15 18:27. In comparison to that, Fig. 3c shows the plot for cases associated with product 1 (an arbitrary product for which no drift was simulated), which is a completely straight line equal to 1, as expected. We can conclude that for this log the hidden concept drift that was simulated is clearly identified by the algorithm, thus an indication of contextual association induced by the product object is obtained.

The results obtained for Log 2 show no control-flow concept drift. Time drift analysis is visualized in Fig. 4 as median time plots, with sub-log index (along time) on the X axis and the median duration of the *Check product availability* activity on the Y axis. The plot of the full log (a) contains small fluctuations, but has no significant peaks. Thus, we can determine that there is no time drift in this activity throughout the full log. In contrast, the plot of the cases related to product 3 (b) show a clear peak, which was indicated by the algorithm. The peak, marked in the green oval, lasted four sublogs, with a median duration of approximately 75 h, after which the median returned to be approximately 5 h. As it is only observed for the cases of product 3, it is considered a hidden time drift for these cases. All the sub-logs that form the peak contain cases that were executed in March, when Scenario 2 was simulated. Hence, the detection approach succeeded to detect the simulated time drift, as well as the time it occurred in our simulation. For comparison, the median time plot for the cases related to product 2 is shown in Fig. 4c, showing no peaks and hence no hidden time drift.

The examination of Log 3 shows no drifts for the full log, and positive indications of both a hidden concept drift in the control flow for cases associated with Product 3, and a hidden time drift in the duration of the Check product availability for cases associated with Product 2. The drift plots for the entire log, concept drift plot for product 3, and time drift plot for product 2 (focused on Check product availability) are presented in Fig. 5. As can be seen in Fig. 5a and c, although the plots of the full log contain some change of control flow behavior and small fluctuations in the durations of the relevant activity, there are no significant troughs and peaks. Thus, we can determine that there is no concept drift in the full log, and no time drift in the Check product availability activity throughout the full log. In contrast, Fig. 5b and d clearly indicate (hidden) concept and time drifts in cases associated with products 3 and 2, respectively. The indices where the drifts are spotted match the time periods of the simulated contextual association scenarios.

To summarize, the controlled evaluation showed that different scenarios that stem from contextual association could be detected as hidden concept and time drift by the algorithm when one or more behaviors of this kind are manifested in the log.

5.2.2 Empirical evaluation using real-life logs

The suggested approach was further evaluated by application to publicly-available real-life logs. The aim of this evaluation was to test the applicability and establish an ability to detect contextual case association, where no information is available about whether such association exists in a given process. We note that for obtaining clear results we decided to select logs where no concept drift and no time drift were detected for the full log.

With these considerations, we selected two real-life logs to which the algorithm was applied. One log was of a problem management system, which served for the 2013 BPI challenge, and the other was a loan application process in a Dutch financial institute, which served for the 2012 BPI challenge.

BPI Challenge 2013[3]—problem management system—the main purpose of the process is to manage problems reported to a help desk, identify the root cause of each problem and secure the resolution. The log contains 1486 cases and 6660 events, with 7 different activity types performed by 587 different users.

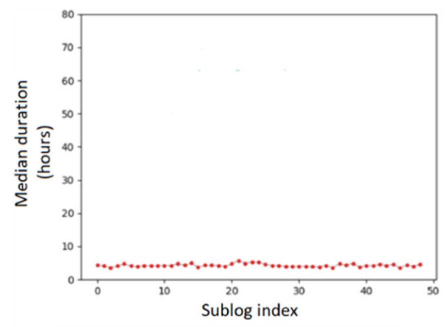
In order to identify possible shared objects which might induce contextual association in the problem management process, we rely on the log and process background provided by the process owner, as well as on our knowledge of the technical problems domain. Some examples of log attributes that represent shared objects that may induce contextual association are Resource, Organization involved, and Product. We focused on the attribute Org: group, which specifies the business area of the user reporting the problem to the helpdesk, and has 15 values representing 15 different business areas. Considering this as a contextual object, we focus on reporting the findings related to control flow concept drift.

Findings: As noted, the absence of concept drift in the full log was checked before the detection of a hidden drift. The output of the concept drift plugin for the full log is presented in Fig. 6. The parameters in this investigation were determined similarly to the previous ones, which, for the full log were min $= 50$, max $= 200$, step $= 10$, and p-value of 0.1. Although Fig. 6 has some fluctuations in the output graph, they are insignificant according to the selected$p$-value. We therefore conclude there are no drift points in the log.

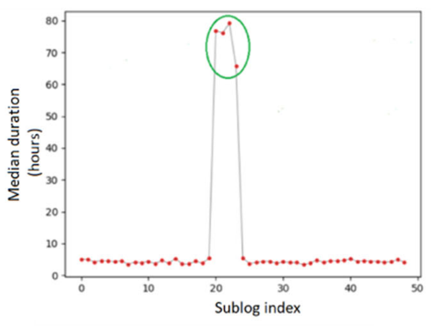
Applying our algorithm to the log, a hidden concept drift was identified for the cases where Org: group is Org line G3, as shown in Fig. 7. These cases form 15% of all cases in the log, and the parameters used by the plugin were set accordingly. The algorithm identified a significant concept drift on index $= 160$ that corresponds to caseID $=$ 1–535,233,659, which started on 2011/06/20 20:30.

For comparison, Fig. 8 shows the concept drift plots obtained for cases where the values of Org:group are Org line C (40% of the cases in the log) and Org line A2 (20% of the cases in the log). The plots indicate that there is no drift for these values, thus we conclude that the drift observed for Org line G3 is potentially induced by a change in the

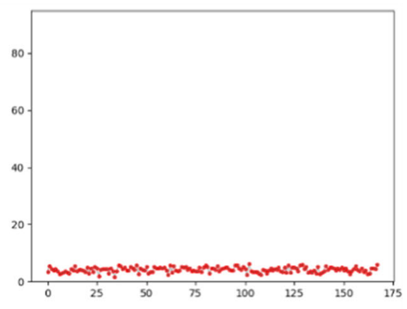
[3] https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07.



(a) Time plot of log 2 (full log)

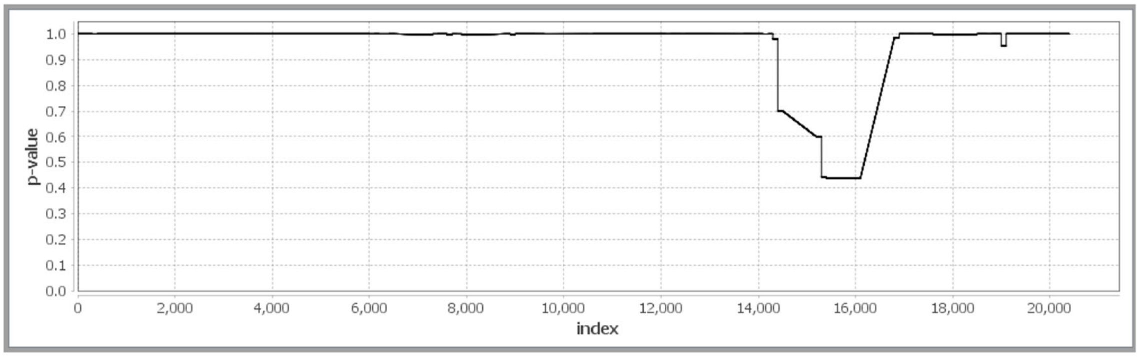


(b) Time plot of log 2 (product 3)

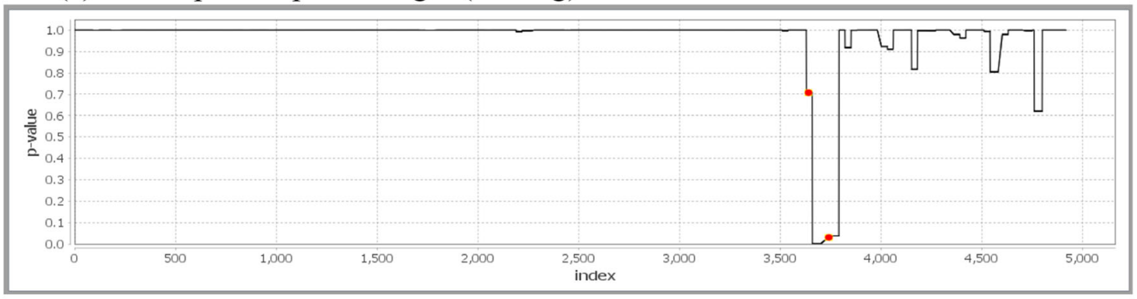


(c) Time plot of log 2 (product 2)

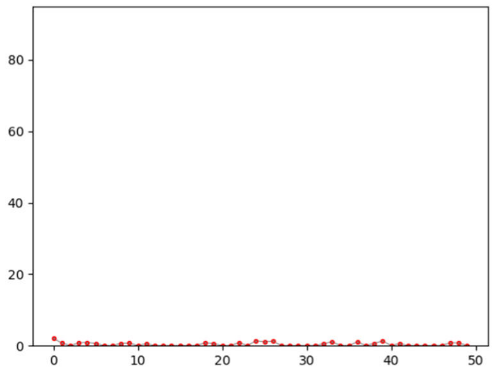
Fig. 4 Median durations of activity *Check product availability* in log 2



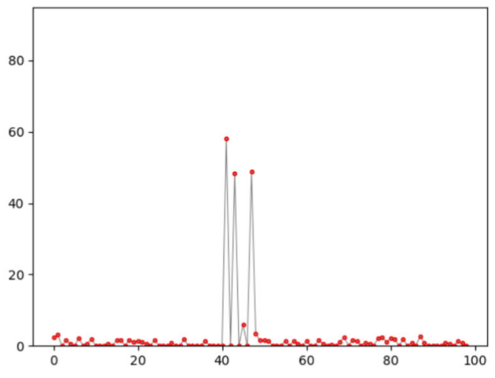
(a) Concept drift plot of log 3 (full log)



(b) Concept drift plot of log 3 (product 3)



(c) Time plot of log 3 (full log)



(d) Time plot of log 3 (product 2)

Fig. 5 Concept drift and time drift plots for log 3

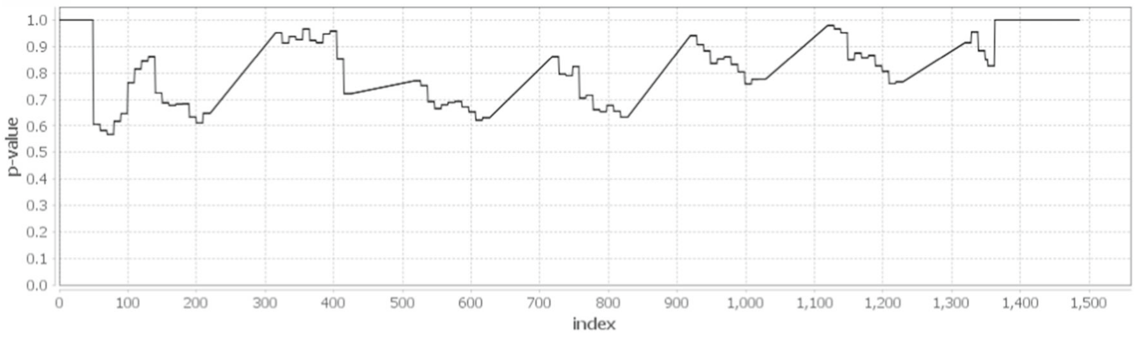


Fig. 6 Concept drift plot of the problem management log

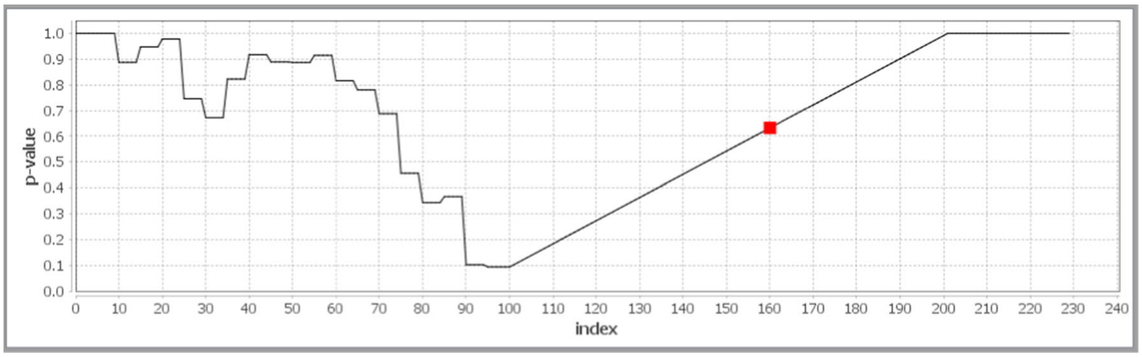


Fig. 7 Concept drift plot for cases where org:group = 'Org line G3'

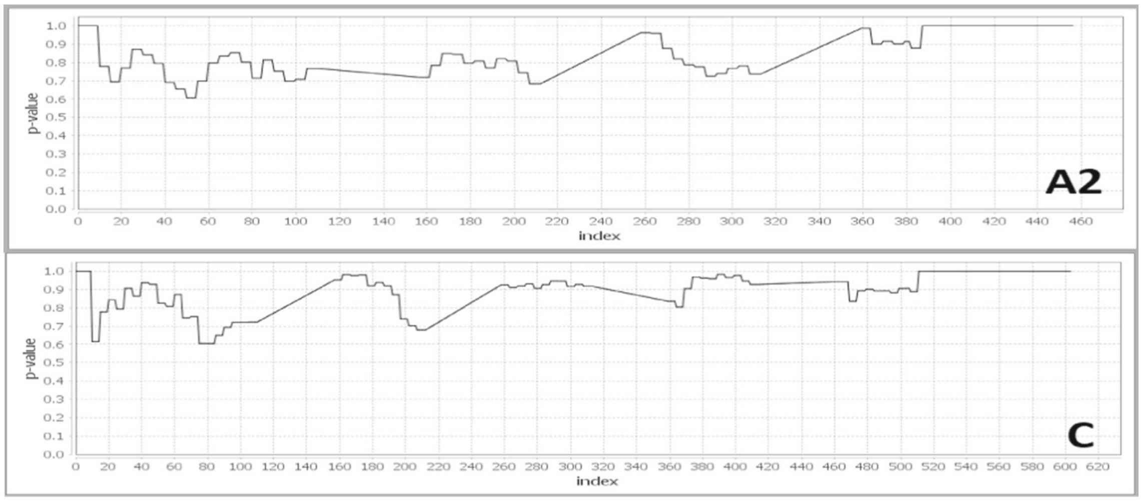


Fig. 8 Concept drift plots for cases where org:group = Org line A2 and Org line C

state of this specific business area, namely, an indication of a contextual association.

To qualitatively assess the contextual association that appears to be induced by the Org:group object in this event log, we manually compared process models discovered for cases related to Org line G3, corresponding to time ranges of before and after the identified drift point. We used divided the original log into two sub-logs, such that one starts at the drift point time, and the second ends at this point. The process models discovered for these sub-logs are presented in Figs. 9 and 10, respectively. The two generated models are different, as a new activity ('accepted wait') is added to the model after the drift. As the behavioral drift only occurs for Org line G3 cases, it can be concluded that the business area induces a contextual association among cases.

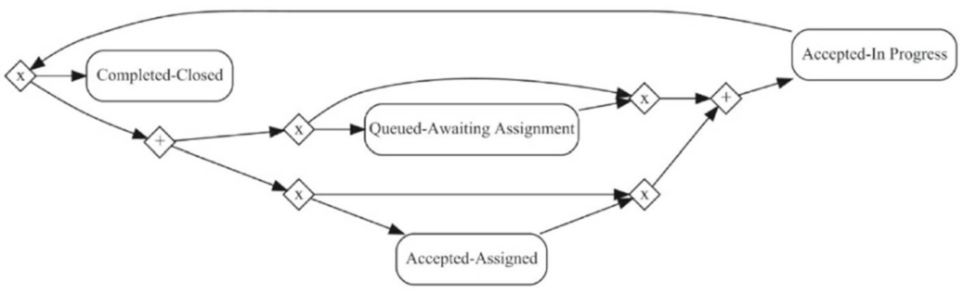


Fig. 9 Process model (BPMN) of the Org line G3 cases, prior to the detected concept drift

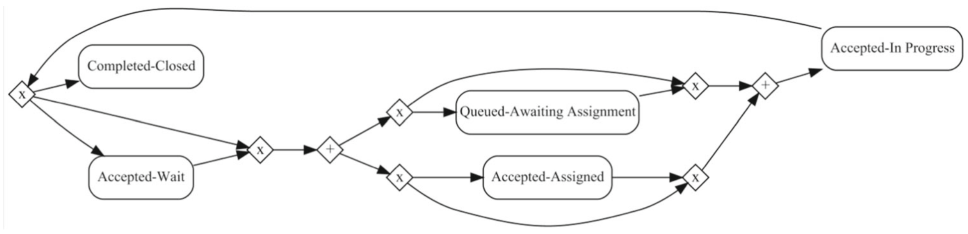


Fig. 10 Process model (BPMN) of the Org line G3 cases, after the detected concept drift

BPI Challenge 2012[4]—a loan application process in a Dutch financial institute—the main purpose of the process is to manage loan applications made by customers, to make and negotiate offers, and to grant the loans eventually. The log contains 13,087 cases and 262,200 events, with 36 activity types performed by 69 different users. The records in the log range over a period from 1-Oct-2011 to 14-Mar-2012, and active cases are distributed uniformly over time.

The shared object we examined in this investigation was the resource (corresponding to the first pattern in Table 3, Sect. 2, while the previous experiments correspond to the second pattern in that table). In this log, each resource handles a great volume of cases, and it is not unlikely in any business process that changes in the state of a resource may affect its performance. For the results of applying our approach to this log, we focus on time drift detection, concerning the activity labeled A_finalized-complete. This activity is included in the execution of most cases in the log, and is performed by various resources.

Findings: As noted, the absence of a time drift in the full log was checked before the detection of a hidden drift. Figure 11 shows the median duration plot of the A_finalized-complete activity for the full log, using a sub-log size of 250. The median durations presented in the plot form an almost straight line close to the value of zero, with a small increase at the end of the log, which can be the result of disrupted and faulty data of the final cases in the log. Thus, we can determine that there is no durational drift in the A_finalized-complete activity throughout the log, and continue to further examine cases associated with specific resources separately.

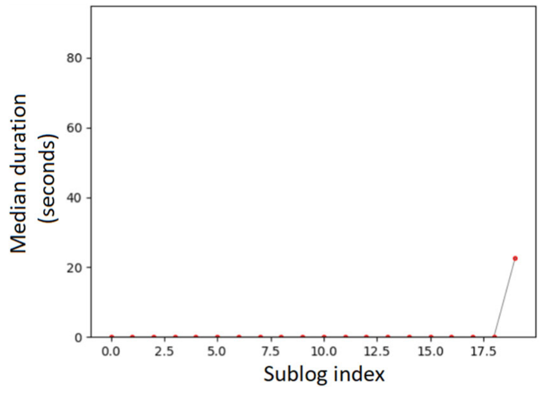


Fig. 11 Median durations of the A_finalized-complete activity, loan application process

Applying our algorithm to this log resulted in identifying a hidden time drift for resource 11,169 in the activity A_finalized-complete, as can be seen in the median duration plot in Fig. 12. The 11,169 resource is the most common resource in the entire log. For comparison, Fig. 13 shows the median duration plots obtained for resources 11,189 and 11,122 (2nd and 3rd most common resources, respectively). The plots lack observable drifts, and this strengthens our premise that a potential cause for the drift detected for resource 11,169 is a change in the state of the discussed resource, namely, a contextual association among cases is induced by the resource.

In summary, applying the method to two real-life event logs has shown that our approach is capable of handling the scale and complexity of such logs and detecting contextual association of cases, which may be manifested either in the control flow or in the time durations of activities in the process. As noted, such results provide indication of possible contextual associations.

4 https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.

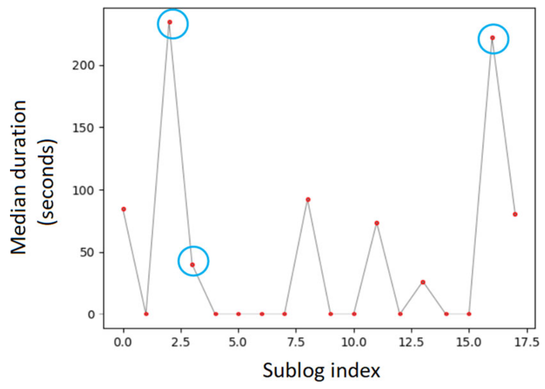


Fig. 12 Time plot (median duration for sublog index along time) of the A_finalized-complete activity performed by resource 11,169; identified time drifts marked by blue circles

6 Related work

While business process management, as well as process mining, primarily focus on within-case behavior, some attempts have been made in recent years to capture, analyze, and mine cross-case behavior as well.

Instance-spanning constraints (ISC), which refer to constraints that span process instances (i.e., cases), have been addressed by a number of works. Formalization of such constraints using event calculus was proposed by Fdhila et al. [17], who also suggested a classification framework applied to the same set of examples we use in this paper [7]. The classification they propose is based on properties such as the domain, sources, and usage of ISC, as opposed to the classification we propose here, which targets behavioral patterns to facilitate their detection. Discovery or detection techniques of ISC have been proposed by Winter et al. [3, 9], addressing specific constraint types which correspond to some of our intentional association patterns. A more generic approach for ISC detection based on SQL queries has been proposed by Aamer et al. [9]. The queries, however, need to be formulated specifically for known and well-defined behaviors. ISC have also been addressed from a compliance management perspective [18]. We note that the notion of ISC is somewhat different than our notion of cross-case association patterns. Specifically, ISC refer both to cross case and cross-process constraints, while our notion refers only to association among cases of the same process type. We believe that a narrower concept can support a more focused investigation. Furthermore, ISC refer to constraints that hold over processes, and hence they capture only normative behavior. In contrast, our cross-case association can be normative or in violation of rules (as is the split-case workaround). A more fundamental view of cross-case association is suggested by Steinau et al. [19], who advocate process relation awareness, as a means for capturing many-to-many relations among processes, which, in turn, create associations among cases of each process through shared data objects. While this notion is fundamental, and also supported at runtime, it is limited to associations through data sharing, and does not capture all the patterns we indicate.

A common cross-case behavior which has recently received attention is batching behavior, where several cases synchronize and are processed together in one process activity or more. An early discovery approach of batching behavior has been suggested by Wen et al. [20]. More recently, detection of such behavior was addressed using different methods [21], including causal event models [22] and performance spectrum visualization [23]. The latter has also served as a basis for incorporating batch-related features for time prediction [14].

Another common cross-case association stems from shared resources of limited capacity, with queues that may form in front of the resources, where cases are delayed (in correlation with one another), and are possibly prioritized according to business rules. Discovery and analysis of queueing behavior based on event logs has been proposed

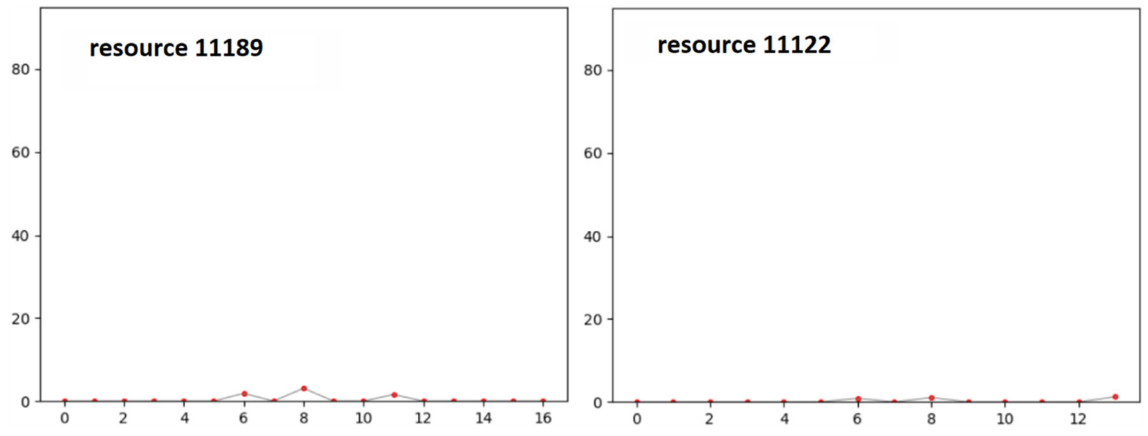


Fig. 13 Median durations of the A_finalized-complete activity performed by resources 11,189 and 11,122

in the form of queue mining [24], and its consideration for time prediction [25]. An additional way in which a limited resource capacity manifests itself is in resource allocation rules, which, again, create association among process cases. Devising such rules based on event logs has been proposed by Arias et al. [26]. According to our classification, behavior that stems from a shared resource is part of the contextual association. We, however, do not address specific patterns such as queuing and prioritization, but rather focus on identifying the existence of association through hidden drifts. We note that changes in the availability of a shared resources can result in a hidden time drift and thus be indirectly discovered by our approach.

A general area where cross-case association has started to be considered is predictive analysis of processes, where these associations are taken into account as inter-case properties or features. This has mainly been considered for the purpose of predicting the remaining time to completion of a case [5, 14, 27], and to a lesser extent for other prediction tasks (e.g., process outcome variables [28]). All these approaches rely on associations that are assumed to be known and formalized into the inter-case features, and are not dedicated to discovering the associations.

7 Discussion

While some attention has already been given to cross-case behavior, the framework suggested in this paper forms a step towards addressing such behavior in a systematic manner. We now discuss how this framework can guide the development of techniques for a variety of purposes. For discovery of cross-case patterns, the distinction made here between intentional and contextual association bears important consequences. In particular, it implies that while additional discovery techniques for intended association should rely on conceptualizations of such patterns, discovery techniques of contextual associations will require generic approaches. The approach proposed here for identifying contextual association depends on the existence of drifts, and is hence only capable of detecting a part of such situations. Additional approaches which would detect correlation among cases that share contextual aspects along time can complement our proposed approach.

Conformance checking and compliance analysis can consider constraints that refer to the patterns (also termed ISC [17]). We note that while such constraints can only refer to intended association patterns (where a clear conceptualization is possible), detecting contextual association can inspire the formation of specific constraints over cases that share context. In addition, compliance analysis can refer to behaviors such as the split-case workaround, which are not possible to detect otherwise.

For prediction purposes, inter-case features can be devised, concerning both intended and contextual association, to be considered by prediction approaches.

Last, anticipating associations across cases is vital for handling exceptions in an appropriate manner. The implications for exception handling are explicit concerning the contamination pattern which is part of our framework. Yet, any contextual association can lead to a chain response, where additional cases are affected by an exception in one case due to their shared contextual objects [29]. While possible cross-case effects of exceptions have long been recognized [30], identifying the set of possibly affected cases has been largely neglected. Detection of contextual cross-case association will enable incorporating it into exception handling mechanisms, so the set of affected cases can be addressed in a targeted manner.

We note that this paper takes a case-centric view, addressing XES-based event logs, which are currently prevalent in process mining techniques. Recently, however, object-centric logs have started to emerge (e.g., OCEL [31]). As opposed to case-centric logs, object-centric logs lack the concepts of trace and case ID. Rather, they hold events and objects, where many objects can be manipulated by many events (e.g., an order, a customer, a product, a package). Analysis approaches which have already been proposed for such logs include discovery of directly-follows multi-graphs [32] and object-centric Petri nets [33] for partial views of the log, typically centered around a subset of the objects in the log. This way, object-centric logs facilitate an analysis of the interaction among related processes that center around the different objects in the log. They can also support an easy discovery of some of the patterns we indicate, such as batching behavior, where a single event manipulates many instances of the same object. Yet, most of the cross-case patterns depicted in our framework are not immediately identifiable from object-centric logs, as the analysis refers to interaction among cases of different processes rather than to interaction among different cases of the same process. The framework proposed in this paper can guide the development of techniques for analyzing such patterns in object centric logs, possibly as extensions of the case-centric techniques discussed here.

Although the work presented in this paper is intended to serve as a step towards a systematic handling of cross-case behavior, it also has implication for practice. The specific patterns we selected for our example analysis approaches are patterns whose detection bears important consequences to organizations. The split-case workaround is a non-compliant behavior which is well known to exist in practice [11]. Bypassing required controls exposes organizations to risks and may even be associated with fraud. Proposing a method for detecting this behavior is, therefore, a valuable contribution for practice. As this is a workaround behavior, its detection can also serve as a basis for process improvements, as suggested by Outmazgin and Soffer [34]. The identification of contextual association among cases is also of considerable importance. Awareness of such an association can contribute to the formulation of inter-case features for prediction tasks [5]. It can also help in analyzing the root causes of delays and performance issues in the process and guide process improvement directions. Furthermore, awareness of such cross-case associations may be essential upon exceptions in the process that may trigger responses [28] and lead to additional exceptions in other cases due to their contextual association.

Several limitations of the work need to be acknowledged. One limitation concerns the proposed framework of association patterns, whose completeness cannot be guaranteed. It relies on an extensive set of examples, extracted from literature, from projects, and from our own experience. Yet, additional patterns may be revealed in future. Nevertheless, the distinction between intended and contextual associations is fundamental. Hence, additional patterns that may still be identified will likely fall into one of these two categories.

Another limitation is that the proposed approach for detecting contextual association may reveal traces of existing associations, but may miss others. As noted, it depends on the existence of drifts over time, and is by no means a comprehensive detection approach. Yet, the generic nature of this approach, which is independent of a particular manifestation, makes it valuable in many different situations.

Last, a limitation which became apparent through our evaluation with real-life logs, is that both our detection approaches indicate "suspicious" behavior, that is not necessarily a reflection of actual association. In practice, such indications need further examination and investigation by the process owner to assess whether the suspected behavior indeed takes place. Nevertheless, this information is essential for the process owner to initiate such investigation, as there are no other techniques that support such detection.

While future research may address the limitations of the current work by identifying additional association patterns to extend the framework and developing additional and complementary techniques for detecting cross-case associations, we believe that full certainty about the detection results will not be achieved, and they will inherently be indications that trigger additional investigations in the organizations.

8 Conclusion

This paper takes a broad look at cross-case behavior, which is frequently overlooked in the context of business processes. We presented a framework of such behaviors, specifically distinguishing intended association patterns and contextual association ones. This distinction bears consequences for how they can be detected in event logs. We demonstrated this by analyzing and proposing detection approaches for an example pattern of each type.

Intended patterns of cross-case behavior are patterns where the association among cases is created intentionally by process participants. They are explicit and can hence be conceptualized precisely and detected accordingly in an event log, as we demonstrated for the split-case workaround. We note that being based on such conceptualizations, the detection of intended association behaviors is pattern-specific and does not encompass different patterns together. In contrast, for contextual association patterns the actual manifestation of the association is not known a-priori. Identifying traces of this association in the form of hidden drifts may apply to a variety of manifestations, all stemming from the relation to a contextual object.

A main contribution of this paper is setting a basis for further systematic development of techniques that will form a toolset for identifying cross-case behaviors and utilizing them for various purposes, such as evolving and improving processes, assessing compliance with regulations and constraints, making predictions, and handling exceptions. We intend, accordingly, to extend the toolset for cross-case association detection by addressing additional patterns and developing additional detection mechanisms.

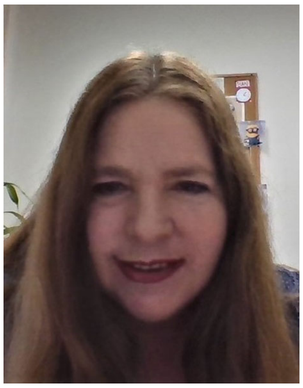
Acknowledgements The research was supported by the Israel Science Foundation under grant agreement 669/17.

References

1. Verbeek, H.M.W., Buijs, J.C.A.M., Dongen, B.F., Aalst, W.M.P.: XES, XESame, and ProM 6. Inform. Syst. Evolut.**72**, 60–75 (2011)
2. van der Aalst W.M.P. et al., (2011). Process mining manifesto. In *International conference on business process management* (pp. 169–194). Springer, Berlin, Heidelberg.
3. Winter, K., Rinderle-Ma, S.: Discovering instance-spanning constraints from process execution logs based on classification techniques. In: *Enterprise Distributed Object Computing Conference*, pp. 79–88 (2017).
4. Kannan, K.S., Manoj, K., Arumugam, S.: Labeling methods for identifying outliers. Int. J. Stat. Syst. **10**(2), 231–238 (2015)
5. Grinvald, A., Soffer, P., & Mokryn, O.: Inter-case properties and process variant considerations in time prediction: A conceptual framework. In: Enterprise, Business-Process and Information Systems Modeling, pp. 96–111. Springer, Cham (2021).
6. Dubinsky, Y., Soffer, P.: Detecting the "Split-Cases" Workaround in event logs. In: Enterprise, Business-Process and Information Systems Modeling, pp. 47–61. Springer, Cham (2021).
7. Rinderle-Ma, S., Gall, M., Fdhila, W., Mangler, J., Indiono, C.: Collecting examples for instance-spanning constraints (2016). http://arxiv.org/abs/1603.01523
8. Aamer, H., Montali, M., Bussche, J. V. D.: What can database query processing do for instance-spanning constraints?. *BPM 2022 workshops* (2022).
9. Winter, K., Stertz, F., Rinderle-Ma, S.: Discovering instance and process spanning constraints from process execution logs. Inf. Syst. **89**, 101484 (2020)

10. van der Aalst W.M.P., et al.: ProM: The process mining toolkit. BPM (Demos) **489**(31), 2 (2009)
11. Outmazgin, N., Soffer, P.: A process mining-based analysis of business process work-arounds. Softw. Syst. Model. **15**(2), 309–323 (2016)
12. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance checking. Springer, Switzerland (2018)
13. De Leoni, M., van der Aalst, W.M.P.: Data-aware process mining: discovering decisions in processes using alignments. In: Proceedings of the 28th annual ACM symposium on applied computing (2013).
14. Klijn, E. L., & Fahland, D. (2020). Identifying and reducing errors in remaining time prediction due to inter-case dynamics. In: 2020 2nd International Conference on Process Mining (ICPM), pp. 25–32. IEEE, New York
15. Kim, J., Jonghyeon K., Suhwan L.: Business process intelligence challenge 2019: Process discovery and deviation analysis of purchase order handling process.
16. Martjushev, J., Bose, R. J. C., Van Der Aalst, W. M.: Change point detection and dealing with gradual and multi-order dynamics in process mining. In: Perspectives in Business Informatics Research: 14th International Conference, BIR 2015, Tartu, Estonia, August 26–28, 2015, Proceedings 14 (pp. 161–178). Springer International Publishing (2015).
17. Fdhila, W., Gall, M., Rinderle-Ma, S., Mangler, J., Indiono, C.: Classification and formalization of instance-spanning constraints in process-driven applications. In: International Conference on Business Process Management, pp. 348–364. Springer, Cham (2016).
18. Amin, R.: Handling instance spanning constraints in compliance management. ABC J. Adv. Res. **8**(2), 95–108 (2019)
19. Steinau, S., Andrews, K., Reichert, M.: The relational process structure. In International Conference on Advanced Information Systems Engineering, pp. 53–67. Springer, Cham (2018).
20. Wen, Y., Chen, Z., Liu, J., Chen, J.: Mining batch processing workflow models from event logs. Concurr. Comput. Pract. Exp. **25**(13), 1928–1942 (2013)
21. Martin, N., Pufahl, L., Mannhardt, F.: Detection of batch activities from event logs. Inf. Syst. **95**, 101642 (2021)
22. Waibel, P., Novak, C., Bala, S., Revoredo, K., Mendling, J.: Analysis of Business Process Batching Using Causal Event Models. In *International Conference on Process Mining*, pp. 17–29. Springer, Cham (2020).
23. Klijn, E. L., Fahland, D.: Performance mining for batch processing using the performance spectrum. In: International Conference on Business Process Management, pp. 172–185. Springer, Cham (2019).
24. Senderovich, A., Leemans, S. J., Harel, S., Gal, A., Mandelbaum, A., van der Aalst, W. M.: Discovering queues from event logs with varying levels of information. In: *International Conference on Business Process Management*, pp. 154–166. Springer, Cham (2016).
25. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Inf. Syst. **53**, 278–295 (2015)
26. Arias, M., Rojas, E., Munoz-Gama, J., Sepúlveda, M.: A framework for recommending resource allocation based on process mining. In: International Conference on Business Process Management, pp. 458–470. Springer, Cham (2016)
27. Pourbafrani, M., Kar, S., Kaiser, S., & van der Aalst, W. M. P.: Remaining time prediction for processes with inter-case dynamics. In: 2nd International Workshop on Leveraging Machine Learning in Process Mining ICPM (2021).
28. Senderovich, A., Di Francescomarino, C., Maggi, F.M.: From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring. Inf. Syst. **84**, 255–264 (2019)
29. Tsoury, A., Soffer, P., Reinhartz-Berger, I.: How well did it recover? Impact-aware conformance checking. Computing **103**(1), 3–27 (2021)
30. Russell, N., Aalst, van der W.M.P., Hofstede, A. T.: Workflow exception patterns. In: International Conference on Advanced Information Systems Engineering, pp. 288–302. Springer, Berlin (2006).
31. Ghahfarokhi, A. F., Park, G., Berti, A., van der Aalst, W.M.P.: OCEL: A standard for object-centric event logs. In: European Conference on Advances in Databases and Information Systems, pp. 169–175. Springer, Cham (2021).
32. van der Aalst W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In *International Conference on Software Engineering and Formal Methods*, pp. 3–25. Springer, Cham (2019).
33. van der Aalst W.M.P., Berti, A.: Discovering object-centric Petri nets. Fundamenta informaticae **175**(1–4), 1–40 (2020)
34. Outmazgin, N., Soffer, P., Hadar, I.: Workarounds in business processes: A goal-based analysis. In: International Conference on Advanced Information Systems Engineering, pp. 368–383. Springer, Cham (2020)

Yael Dubinsky is the head of an information systems development team in a defense technology company in Israel. She received her B.Sc. (2018) in Information Systems Engineering from Ort Braude College in Israel, and M.Sc. in Information Systems from the University of Haifa (2022). This paper is the first paper published based on her M.Sc. Thesis work, in which she studied the discovery of inter-case patterns of behavior using process mining techniques. As part of her professional experience, Yael has led, designed and developed many IT projects focusing on information systems.



Pnina Soffer is a Full Professor and the former Head of the Information Systems Department at the University of Haifa, Israel. Her research deals with business process management, including process mining and process modeling, as well as conceptual modelling in general. She typically combines in her research formal and technical development of analysis approaches with empirical investigation of human aspects concerning the developed technology. She has published over 150 papers in journals and conference proceedings. She has served in editorial boards of various journals (e.g., Decision Support Systems, Journal of the AIS), and is currently an Associate Editor in Data & Knowledge Engineering and a Department Editor of the information systems engineering department of Business & Information Systems Engineering (BISE). She has served in program committees of numerous conferences, and is a member of the steering committee of CAiSE and of the IEEE Taskforce for Process Mining. She held many organizational positions in conferences. In particular, she served as a Program chair in BPM 2014, CAiSE 2016, and ICPM 2021, and a General chair of the upcoming CAiSE 2024.

Irit Hadar is an Associate Professor and the Head of the Department of Information Systems, University of Haifa. Hadar co-founded and heads the interdisciplinary research hub: Deign Thinking for Socio-Technical Innovation, involving members from 15 different research disciplines. Her research focuses on cognitive and social aspects of requirements engineering and software architecture. Hadar serves as an editorial board member of Empirical Software Engineering and Requirements Engineering and has been serving as an organizer and PC member in conferences and workshops (e.g., ICSE, RE, EASE, CAiSE, ICIS) and as a Program Chair of the upcoming RE 2024.