Interactive log-delta analysis using multi-range filtering

Maxim Vidgof¹ · Djordje Djurica¹ · Saimir Bala¹ · Jan Mendling¹

Received: 16 November 2020 / Revised: 2 May 2021 / Accepted: 7 June 2021 / Published online: 14 September 2021
© The Author(s) 2021

Abstract

Process mining is a family of analytical techniques that extract insights from an event log and present them to an analyst. A key analysis task is to understand the distinctive features of different variants of the process and their impact on process performance. Techniques for log-delta analysis (or variant analysis) put a strong emphasis on automatically extracting explanations for differences between variants. A weakness of them is, however, their limited support for interactively exploring the dividing line between typical and atypical behavior. In this paper, we address this research gap by developing and evaluating an interactive technique for log-delta analysis, which we call *InterLog*. This technique is developed based on the idea that the analyst can interactively define filter ranges and that these filters are used to partition the log L into sub-logs L_1 for the selected cases and L_2 for the deselected cases. In this way, the analyst can step-by-step explore the log and manually separate the typical behavior from the atypical. We prototypically implement *InterLog* and demonstrate its application for a real-world event log. Furthermore, we evaluate it in a preliminary design study with process mining experts for usefulness and ease of use.

Keywords Process mining · Log-delta analysis · Variant analysis · Multi-range filter · Event logs · Event sequence data

1 Introduction

Business process management (BPM) is concerned with the analysis and redesign of organizational processes [14]. Event sequence data from process-aware information systems are increasingly available as event logs to support BPM. Various techniques have been developed for extracting actionable insights from such event logs, including automated process discovery, conformance checking, variant analysis and performance analysis. All these techniques are specific process mining techniques [27].

One specific challenge for process mining techniques is the effective distinction between typical and atypical behavior. Process discovery techniques considering the overall event log often produce large spaghetti-like models and models having either low level of fitness to the event log or low precision or generalization [27]. There are largely two approaches to tackle this problem: first, by automatically

eliminating such behavior that is considered to be noise based on some thresholds [8] and second, by providing the analyst with some support for interactively filtering the log [29]. A recent review illustrates that even the arguably best process discovery algorithms are challenged by the complexity of real-world event logs if they are not filtered or preprocessed [1]. Therefore, it is an open research question how analysts can be best supported to interactively explore the dividing line between typical and atypical behavior of an event log.

In this paper, we address this research gap by developing and evaluating an interactive technique for log-delta analysis, which we call *InterLog*. This technique is developed based on the idea that the analyst can interactively define filter ranges and that these filters are used to partition the log L into a sub-log L_1 for the selected cases and another sub-log L_2 for the deselected cases. In this way, the analyst can step-by-step explore the log and manually separate the typical behavior from the atypical. We prototypically implement the *InterLog* and demonstrate its application for a real-world event log. Furthermore, we evaluate it in a preliminary design study with process mining experts for usefulness and ease of use.

This paper is structured as follows. Section 2 describes the practical problem that our technique addresses and the associated research gap with a focus on five design requirements.

Communicated by Selmin Nurcan and Pnina Soffer.

✉ Maxim Vidgof
maxim.vidgof@wu.ac.at

¹ Institute for Information Business, Vienna University of Economics and Business (WU), Vienna, Austria

Section 3 describes the design of the *InterLog* technique from a conceptual perspective. Section 4 presents our prototypical implementation and its application to a real-world event log. Furthermore, we report the results of a design study with process mining experts. Section 5 discusses the benefits of the *InterLog* technique with respect to the previously defined requirements and the overall research question. Finally, Section 6 concludes the paper and discusses future work.

2 Theoretical background

This section describes the problem and provides an analysis of related literature. Based on this analysis, we identify five requirements for an interactive log-delta analysis technique.

2.1 Problem description

One of the key challenges for a process analyst is to understand what is the typical and the atypical behavior of a business process. General techniques for automatic process discovery have been struggling to address this challenge. Event logs in practice are complex, often including tens of thousands of cases with millions of events [1]. This complexity of the event log usually translates into complexity of the process models that are automatically generated by process mining tools. Such overly complex models are also referred to as Spaghetti models, because it is hard to identify specific paths in their dense and chaotic graph representation.

Let us illustrate this problem by the help of an example. Figure 1 shows a simple complaint handling process adapted from [14]. The process works as follows. After clients file a complaint, they immediately receive an automated confirmation message. Next, an employee brings the application to a meeting with colleagues in order to discuss a solution. The same employee is in charge of contacting back the customer with an apology and proposing a solution. The solution may be accepted or rejected by the client. In case of acceptance, the solution is executed right away. In case of rejection, the employee contacts the client to investigate on alternatives. As long as a reasonable alternative is found, the employee has a new meeting with colleagues to discuss the solution and proceed as usual. If no alternative solutions can be found, the complaint is brought to court and the process fails.

There are several ways in which instances of the process may traverse the depicted process model. The typical behavior of the process is the sunny-day scenario, and it is the one in which an agreement with the client is found right away. In a good process this case should occur frequently. In contrast, the rainy-day scenario relates to cases that do not lead to an agreement and the company is brought to court. In this case, the costs sustained from the company may be much higher than settling for a solution. An intermediate scenario is the

one in which a customer does not accept the first proposed solution, but some iterations are done.

The company is now interested to better understand the dividing line between the typical and the atypical behavior of the process. To this end, an analyst is assigned the task of conducting a process mining project in order to identify potential for improvement. Table 1a lists the activities of the process with their corresponding short label. Process mining tools provide a first overview by listing variants as shows in Table 1b (here sorted by frequency). Each variant describes one specific sequence through the process. Process mining tools offer analysts facilities to interactively filter the event log. However, applying such filters does not directly provide an answer to the question how typical and atypical behavior is different. So the analyst resorts to the strategy of applying different filters for creating separate models for groups of variants.

2.2 Prior research on identifying typical and atypical behavior

In general, there are two groups of techniques that help addressing the described problem: techniques based on filtering and log-delta analysis techniques.

2.2.1 Filtering techniques

Filtering an event log can be achieved with the use of three types of filters that remove a subset of the traces, events or event pairs intending to produce a simpler log. Event filters allow users to remove or to keep all the events that satisfy a predefined condition set by the user. They allow users to focus only on a particular activity. Event pair filter allows users to remove or keep all the pairs of events that fulfill a specific condition. This type of event log filtering is used to show a relation between two events and gather more insight into, for instance, situations where event A is followed by event B. Finally, using trace filters enables users to remove or retain all the traces from the log that fit the defined criteria. This filter can be used to, for example, show all the traces that occur with a defined level of frequency or all traces that have a specific duration of cycle time [15].

Conforti et al. [9] discuss filtering for excluding infrequent behavior from event logs. They distinguish the following filter operations. First, filter log by attribute removes all the events where the value of the attribute is not equal to the value defined by the user. It can also remove all the events that do not contain a certain selected attribute. Second, filter on time frame serves to filter out all the events which fall into the desired time frame. Some filters are refind to filter out atypical behavior. The filter log using simple heuristics removes all the traces that do not start or end with a particular event. It also can remove all the events related to the specific

Fig. 1 Example of a complaint handling process (adapted from [14])

Table 1 Process activities and variants

a Activities of the process	
Activity or event	Label
Complaint received	A
Send automatic reply to customer	B
Discuss solution	C
Send apology	D
Propose solution	E
Positive response received	F
Execute solution	G
Complaint addressed	H
Negative response received	I
Evaluate acceptable alternative	J
Go to court	K
Complaint not addressed	L
b Process variants ordered by trace frequency	
Variant	Count
$\langle A, B, C, D, E, F, G, H \rangle$	807
$\langle A, B, C, D, E, I, J, C, D, E, F, G, H \rangle$	132
$\langle A, B, C, D, E, I, J, K, L \rangle$	30
$\langle A, B, C, D, E, I, J, C, D, E, I, J, C, D, E, F, G, H \rangle$	21
$\langle A, B, C, D, E, I, J, C, D, E, I, K, L \rangle$	6
$\langle A, B, C, D, E, I, J, C, D, E, I, J, C, D, E, I, J, C, D, E, F, G, H \rangle$	2
$\langle A, B, C, D, E, I, J, C, D, E, I, J, C, D, E, I, J, K, L \rangle$	2

process task by calculating frequencies of event occurrence. Furthermore, the filter log using prefix-close language eliminates all the traces that are not a prefix of another prefix in the log by using a frequency threshold defined by the user.

A second group of techniques approaches the distinction between typical and atypical behavior as a clustering problem [23,30]. Trace clustering is a technique where the event log is divided into homogeneous subsets which are then used to create separate process models. This approach is able to cope with real flexible environments and improve process mining results. A prominent example designed for process mining is active trace clustering [30] inspired by principles of active learning. This approach borrows elements from machine learning and utilizes a selective sampling strategy which enables an active learner to decide which instances to

select based on their informativeness. The most frequently used informativeness measure is the frequency of the trace.

A third group of techniques are discovery algorithms that interpret infrequent behavior as noise. The most well-known ones are Heuristics Miner [31], Inductive Miner [17] and Fuzzy Miner [15]. The Heuristics Miner deals with noise by introducing frequency-based metrics, while Inductive Miner uses two types of filters that accomplish this. The first filter applies a similar approach to Heuristics Miner and removes all the edges from the directly follows graphs. In contrast, the second filter removes edges that the first filter did not remove by using eventually follows graphs. However, process models mined using Inductive Miner are often oversimplified. A different approach to the previous two is Fuzzy Miner. This algorithm filters noise directly on the discovered model using

the desired level of significance and correlation thresholds defined by users.

As we can see, there are numerous techniques and algorithms which can be used to simplify event logs and models to help users understand the core process better. However, all of them are achieving this by filtering out atypical behavior, considering it a behavior to be neglected [27]. We argue that this is a substantial limitation that needs to be addressed since atypical behavior can carry important information which is lost by eliminating it from the log. Obtaining insights into rare cases can help companies detect errors in the process or even detect fraud. Furthermore, none of the presented techniques considers that users might want to observe a process model that comprises both the most frequent and infrequent traces of the process.

2.2.2 Log-delta analysis

There are various techniques that belong to the family of log-delta analysis, which is also referred to as variant analysis [25]. A specific approach in this category is deviance mining, which is used for detecting and explaining differences between process executions that produce positive outcomes compared to executions that produce negative ones [26]. Deviance mining splits a log file into cases with satisfactory performance ($L1$) and undesirable performance ($L2$) based on specific criteria. This type of analysis can be done both manually and automatically.

The manual approach builds on the process model discovery for $L1$ and $L2$. Once the models are discovered, users can compare them visually. [14] provide several examples of visual model comparison, where users can spot differences in frequency (differences in activity frequencies in $L1$ and $L2$) and performance views (differences in cycle times). An approach following this principle is presented in [16], focusing on mining clinical care pathways and their correlation with patient outcomes. Their work is realized as a set of interactive tools on the business process insight (BPI) platform. It enables users to interactively use individual clustering, process mining and frequent mining capabilities. However, this manual approach is error-prone and suffers from increasingly complex event logs.

The issue of complexity is addressed by log-delta analysis techniques that capture a set of patterns that are common for $L1$ and uncommon for $L2$, or vice versa. Users can then analyze these patterns and discover those that explain the observed differences in performance. Specific techniques in this category include discriminative sequence mining techniques and techniques based on association rules [14]. Discriminative sequence mining techniques take two logs as input and produce patterns based on their discriminative power. This means that a pattern is selected if it is observed only in one log. An example of such approach is

[18]. Here, the authors focus on classifying software reliability issues using a classifier based on run history, which enables them to capture failures and atypical behavior and generalize previously known errors. The approach builds on mining discriminative features, followed by capturing the program execution traces' recurrent series of events. Once this is completed, feature selection is performed, and the best features are chosen and used to train a classifier to observe failures. Techniques based on association rules use one log as an input and produce frequent patterns from the set of positive and negative cases separately. An example of such pattern is that after activity A, we will eventually observe activity B [14].

There are also various contributions to log-delta analysis. Van Beest et al. [26] present an approach to log-delta analysis in which users can distinguish between normal and deviant executions of the process or between two variants of the same process by using a lossless encoding of an event log. The event log is encoded as an event structure and combined with a frequency-enhanced technique for differing pairs of event logs in their technique. Cordes et al. [10] offer a tool that can discover two process models, spot and visually emphasize the differences between them. This work has been further extended to include a comparison between process variants using annotated transition systems [2]. The work by Bolt et al. [4] uses transition systems for modeling the behavior of the process variants. Then, differences are highlighted and dominant behavior is presented. The states on the transition system are interactive, and when clicked, they provide further details on the variant differences to the user. The authors state that this approach is capable of detecting relevant differences but avoids detecting insignificant ones. Their work was further extended to include decision trees for each decision point of the transition system [5]. Research by Taymouri et al. [24] presents an approach for detecting statistically significant differences between process model variants based on the so-called mutual fingerprint. The authors define mutual fingerprint as a graph created from the event logs of two variants, enabling lossless encoding of trace sets and their duration, consequently enabling the tool to capture statistical differences in the control flow and performance dimensions. The evaluation shows that this approach can detect spurious differences but cannot always detect baseline differences at a trace level. Recent research by Boltenhagen et al. [6] formulates variant analysis as a constraint satisfaction problem using a notion of trace variant that considers concurrency and iteration. The ambition of this work is to balance complexity and quality of identified trace variants.

2.3 Requirements for an interactive log-delta analysis technique

Based on the previously presented research, we note that most of the log-delta techniques, although capable of handling complex logs and automatically comparing discovered process models, suffer from certain limitations. Most of the presented tools lack interactive support. This makes it difficult for analysts to explore and filter the event log for determining how typical and atypical behavior is different based on multiple criteria. This results in analysts having to create separate models for different groups of variants.

In our work, we have observed the need inspect different groups of process variants and observe and manipulate two discovered models simultaneously by changing filtering values. Similar observations have been made in user studies in the field of visual analytics. Du et al. [13] describe a set of 15 strategies that analysts of event sequence data frequently instantiate. Among others, they find that analysts require support for goal-driven record extracting and category extracting in order to identify features that are linked to outcomes. Furthermore, they require support for grouping event categories, analyzing small subset and partitioning. Having such support in place allows users with domain knowledge to interactively explore the process, observing at the same time showing atypical and typical behavior.

Against this background, we formulate the following requirements for an interactive log-delta analysis technique.

RQ1 (Select time range) A filtering technique must allow the user to split the log on the time dimension. That is, first, it must offer a way of selecting which time intervals are included in which of the two sub-logs. Second, it must also offer a way to select which performance intervals are included.

RQ2 (Select traces) A filtering technique must allow the user to split the log on the trace level. That is, it must offer a way of selecting which traces are included in which of the sub-logs. The trace selection must be possible based on the variant dimension as well as on trace attributes.

RQ3 (Select activities) A filtering technique must be able to split the log on the activities dimension. That is, it must offer a way of retaining user-selected activities in one of the sub-logs while including the remaining activities in the other one. The activity selection must be possible on any of the activity attributes, including but not limited to activity label.

RQ4 (Multi-range filtering) A filtering technique must be able to perform the splits on multiple ranges. That is, it must offer a way of selecting several intervals for each of the splits.

RQ5 (Interactive partitioning) A filtering technique must allow the user to work with it in an interactive manner,

meaning the filtering technique must react to user input in near real time.

Table 2 evaluates our technique as well as the existing approaches against these requirements. The plus sign ('+') means that the requirement is supported by the technique, while an empty space means the technique is missing support of this requirement.

3 A technique for interactive log-delta analysis

In this section, we describe our technique for log-delta analysis that allows to split event logs based on multiple user-defined ranges. We show an overview of the technique, provide the necessary definitions and then describe the technique in detail.

3.1 Overview of the technique

Our technique is summarized in Fig. 2. It takes as input an event log and up to five user-defined multi-ranges. For the time filter, a multi-range is a set of intervals of timestamps. For the custom attribute filter, a multi-range is a set of intervals of numeric values. For the performance filter, a multi-range is a set of intervals of percentiles. For each of the remaining filters, it is a set of intervals of frequencies. The filters based on frequency accept values in the interval from 0 to 1, where [0,0] means, for instance, that we get the least frequent variant or activity, and [0,1] means that we consider all possible behavior. The aforementioned multi-ranges are used, respectively, by five filter types: (i) time filter; (ii) variants filter; (iii) activity filter; (iv) performance filter; and (v) custom attributes filter. The custom attributes filter allows for multi-range filtering on any attribute of the event log. These filters can be used independently or consecutively. In the latter case, their application must follow the order: time filter first, variants filter next, performance filter third, activities filter subsequently and the custom attribute filter last. The output of each filter are two event logs, split according to the filtering criteria. If the filters are chained together, the output of this chain follows a more complex logic, which will be explained in Sect. 3.3. The resulting event logs can be used by any process mining technique to generate process models which allow the user to analyze the data. In addition, directly follows graphs are generated for these event logs by our tool in order to support user interaction.

3.2 Preliminaries

In the following, we define the fundamental concepts used by our approach.

Table 2 Overview of related techniques and coverage of identified requirements

Category	Technique	RQ1	RQ2	RQ3	RQ4	RQ5
Filtering	Filter by attribute		+	+		
	Filter on time frame	+	+	+		
	Filter using heuristics		+	+		
	Filter using prefix-close language		+	+		
	Trace clustering		+	+		
	Active trace clustering		+	+		
	Heuristics Miner		+	+		
	Inductive Miner		+	+		
	Fuzzy Miner		+	+		
	van Beest et al. [26]		+	+		
Log-delta analysis	Lakshmanan [16]		+			+
	Lo et al. [18]		+			
	Cordes et al. [10]		+	+		
	Ballambettu et al. [2]		+			+
	Bolt et al. [4], bolt2018process		+			+
	Taymouri et al. [24]		+			
	Boltenhagen et al. [6]		+			
	Our InterLog technique	+	+	+	+	+

Fig. 2 Overview of the InterLog technique

Definition 1 (Event, activity, timestamp) Let $\mathcal{A}$ be the universe of events. Each event has attributes. Let AN be the set of attribute names and $AN_e \subseteq AN$ be the set of event attribute names. For any event $e \in \mathcal{A}$ and name $n \in AN_e$, $\#_n(e)$ is the value of the attribute n for event e . If the value of the attribute n for event e is not defined, we say $\nexists \#_n(e)$. An activity is a specific attribute of an event, i.e., $\#_{\text{activity}}(e)$ is the activity associated to the event and $\#_{\text{timestamp}}(e)$ is the time when the event occurred.

For example, $\#_{\text{activity}}(e) = \text{'Discuss solution'}$, $\#_{\text{timestamp}}(e) = \text{'2020-10-30 10:37:55'}$.

Definition 2 (Trace, variant, event log) A trace $t = \langle e_1, \dots, e_n \rangle$ is a finite sequence of events. A trace might also have attributes. The set of trace attribute names AN_t is such that $AN_t \subseteq AN$ and $AN_e \cap AN_t = \emptyset$, i.e., the sets AN_t and

AN_e are disjoint. For any trace t and a trace attribute name $n \in AN_t$, $\#_n(t)$ is the value of the attribute n for trace t .

An event log $L \subseteq \{t\}^*$ is a multi-set of traces. A process variant is a subset of traces $V \subseteq L$. Variants group together traces which have similarities to one another and differences to traces in other variants. Thus, each $t \in V$ has multiplicity equal to 1. Note that such definition of variant implies that traces have unique trace identifiers.

An example of trace is $t = \langle a, b, c, d, e, f, g, h \rangle$. An example of log is $L = [\langle a, b, c, d, e, f, g, h \rangle^{20}, \langle a, b, c, d, e, i, j, k, l \rangle^5]$. In this event log, the first trace occurs 20 times whereas the second one occurs 5 times.

Definition 3 (Variant frequency, Activity frequency, Attribute frequency) Variant frequency $vf_L(t)$ is defined as the frequency of the occurrence in L of its constituting traces $t \in V$. Activity frequency $af_L(a)$ is defined as the sum of the num-

ber of times activity a occurs in the event log L . More generally, attribute frequency $af_L(attr, val)$ is defined as the frequency of occurrence in the L of the value val for the attribute $attr$.

For example, given $L = [\langle a, b, c, d, e, f, g, h \rangle^{20}, \langle a, b, c, d, e, i, j, k, l \rangle^5]$, then $vf_L(\langle a, b, c, d, e, f, g, h \rangle) = 20$ and $af_L(a) = 25$.

A filtering technique is a function $f : L \rightarrow (L_1, L_2)$ which partitions an event log L into complementary event logs L_1 and L_2 . Next, we use the given definitions to describe the algorithms used by our technique.

3.3 Log partitioning based on multi-range filtering

Our implementation provides five filters: the timestamp filter, the variants filter, the performance filter, the activities filter and the custom attribute filter. These filters are composable but their application is not commutative, i.e., it has to be performed in strictly defined order. Namely, first the timestamp filter is applied, then the variants filter is applied on the results of the timestamp filter, the performance filter is applied on the variants filter's result, the activities filter is applied on the results of the performance filter, and finally, the attribute filter is applied on the result of the activities filter.

Each filter splits its input log L^n into two logs L_1^{n+1} and L_2^{n+1} , with the former log containing the traces (or variants, or activities) within the input ranges and the latter one containing the rest. The union of the two output logs is the input log. If another filter is chained to it, the latter filter would take L_1^{n+1} as its input.

We are interested in filtering at multiple ranges in the event log. These ranges represent frequencies expressed by the user in the form of sets of intervals. That is, $R = \{[\min_0, \max_0], [\min_1, \max_1], \dots, [\min_n, \max_n]\}$ with $\min_i \leq \max_i$, $i = 0, \dots, n$ signifies that the user want to retain from the log an amount of information that falls into either of the intervals $[\min_0, \max_0]$, $\dots$, $[\min_n, \max_n]$. Such ranges can be applied to filtering on the variants level—referred to as R_v —and filtering on the activities level— R_a . They also can be applied to filtering on the additional attribute—referred to as R_{at} —but only in case the attribute is of string type and so af_L can be applied to it. Since the range boundaries are specified as frequency percentiles, the minimum value of $\min_i$ is 0, and the maximum value of $\max_i$ is 1. We also establish that $[\min, \max]$ means that the boundaries of the interval are included and $(\min, \max)$ means the boundaries are excluded. With this definition we can express the non-overlaps condition on the ranges specified by the user as $\forall i, j \in [0..n] \Rightarrow [\min_i, \max_i] \cap [\min_j, \max_j] = \emptyset$. This is a precondition for applying the corresponding filters. In other words, ranges may share boundaries but they must not overlap.

The ranges for the performance filter— R_p —are similar in their structure (being an array of pairs) as well as the value range (from 0 to 1) and in that the non-overlaps condition applies to them, too. However, the range boundaries represent performance percentiles instead of frequency percentiles. That is, a range $R'_p = \{[0, 0.1]\}$ would represent 10% fastest traces.

If the attribute filter is applied on a numeric attribute, the ranges have the same structure, that is $R_{at} = \{[\min_0, \max_0], [\min_1, \max_1], \dots, [\min_n, \max_n]\}$. However, the $\min_i$ and $\max_i$ in this case are not frequencies but simply numeric values. Similarly, the ranges for the timestamp filter are represented by a set of intervals in form $R_t = \{[start_0, end_0], [start_1, end_1], \dots, [start_n, end_n]\}$ with $i = 0, 1, \dots, n$ and $start_i < end_i$ meaning $start_i$ is referring to an earlier point of time than end_i . The non-overlaps condition applies to the time ranges R_t as well as to the numeric ranges R_{at} .

Our implementation consists of six main blocks. First, the ranges specified by the user for each of the applied filters are checked for overlaps. If the ranges are incorrect, an error is produced and the filtering is not applied.

Second, if the ranges are correct, the timestamp filter can be applied. It takes the input log and splits it into two non-overlapping sub-logs. Algorithm 3.3 describes in more detail how this filter is applied. Note that the timestamp filter is applied on the trace level, meaning a trace is included in L'_1 if all of its events take place within user-defined time boundaries. Otherwise, the trace is included in L'_2 .

Algorithm 1: Filter on timestamp

Input: Event log L . Ranges
 $R_t = \{(start_0, end_0) \dots (start_t, end_t)\}, t \in \mathbb{N}_0$
Result: New event logs $L'_1 \subseteq L$ and $L'_2 \subseteq L$ such that
 $L'_1 \cup L'_2 = L$ and $L'_1 \cap L'_2 = \emptyset$

```

1  $L'_1 \leftarrow []$ ;
2  $L'_2 \leftarrow []$ ;
3 forall the  $(start_n, end_n) \in R_t$  do
4   forall the trace  $\in L$  do
5     if  $\forall event \in trace : start_n \leq \#_{timestamp}(event) \leq end_n$ 
6       then  $L'_1 \leftarrow L'_1 \cup \{trace\}$ ;
7     else
8        $L'_2 \leftarrow L'_2 \cup \{trace\}$ ;
9     end
10  end
11 end

```

Third, the variants filter can be applied. The variants are filtered according to Algorithm 3.3. Note that if the timestamp filter is not applied, the initial log L serves as an input for this filter, otherwise it takes L'_1 .

Next, the performance filter is applied. It sorts the traces left in L'_1 by their throughput time. This performance metric is chosen since it requires only one timestamp per event and thus can be applied to all event logs, while other more sophis-

Algorithm 2: Filter variants

Input: Event log L'_1 . Ranges $R_v = \{(min_0, max_0) \dots (min_m, max_m)\}, m \in \mathbb{N}_0$
Result: New event logs $L''_1 \subseteq L'_1$ and $L''_2 \subseteq L'_1$ such that $L''_1 \cup L''_2 = L'_1$ and $L''_1 \cap L''_2 = \emptyset$

```

1  $variants \leftarrow \forall \text{ variants} \in L'_1;$ 
2  $variants \leftarrow \text{sort } variants \text{ by } vf_{L'_1}(variant);$ 
3  $nr\_variants \leftarrow |variants|;$ 
4  $indices \leftarrow \bigcup_{i=0}^m \{n \in \mathbb{N}_0 | n \in [round(min_i \times nr\_variants), round(max_i \times nr\_variants)]\};$ 
5  $filtered\_variants \leftarrow \bigcup_{i \in indices} variants_i;$ 
6  $L''_1 \leftarrow \forall trace \in L'_1 \cap filtered\_variants;$ 
7  $not\_filtered\_variants \leftarrow variants \setminus filtered\_variants;$ 
8  $L''_2 \leftarrow \forall trace \in L'_1 \cap not\_filtered\_variants;$ 

```

---

ticated metrics such as *cycle time* would limit its applicability. Algorithm 3.3 shows the inner structure of this filter.

Algorithm 3: Filter on performance

Input: Event log L'_1 . Ranges $R_p = \{(min_0, max_0) \dots (min_r, max_r)\}, r \in \mathbb{N}_0$
Result: New event logs $L'''_1 \subseteq L'_1$ and $L'''_2 \subseteq L'_1$ such that $L'''_1 \cup L'''_2 = L'_1$ and $L'''_1 \cap L'''_2 = \emptyset$

```

1  $performances \leftarrow [];$ 
2  $L'''_1, L'''_2 \leftarrow [];$ 
3 forall the  $trace \in L'_1$  do
4    $\#_{throughput}(trace) \leftarrow \max(\bigcup_{e \in trace} \#_{timestamp}(e)) - \min(\bigcup_{e \in trace} \#_{timestamp}(e));$ 
5    $performances \leftarrow performances \cup \#_{throughput}(trace);$ 
6 end
7  $nr\_cases \leftarrow |L'_1|;$ 
8  $performances \leftarrow \text{sort } performances;$ 
9  $indices \leftarrow \bigcup_{i=0}^r \{n \in \mathbb{N}_0 | n \in [round(min_i \times nr\_cases), round(max_i \times nr\_cases)]\};$ 
10  $perf\_subset \leftarrow \bigcup_{i \in indices} performances_i;$ 
11 forall the  $trace \in L'_1$  do
12   if  $\#_{throughput}(trace) \in perf\_subset$  then
13      $L'''_1 \leftarrow L'''_1 \cup trace;$ 
14   else
15      $L'''_2 \leftarrow L'''_2 \cup trace;$ 
16   end
17 end

```

Then, we can apply Algorithm 3.3 on the resulting log. First, it builds a list of activities sorted by their frequency, analogous to Algorithm 3.3. Then, a range filter is applied in the same manner. When the ranges are calculated, we iterate over all traces in the input log and rebuild them in such a way that activities within the user-specified ranges are appended to one trace (*new_trace* in Algorithm 3.3), while the remaining activities build up the other one (*not_new_trace* as opposed to *new_trace*). The newly built trace is appended

Algorithm 4: Filter activities

Input: Event log L'''_1 . Ranges $R_a = \{(min_0, max_0), \dots, (min_p, max_p)\}, p \in \mathbb{N}_0$
Result: New event logs $L''''_1 \subseteq L'''_1$ and $L''''_2 \subseteq L'''_1$ such that $L''''_1 \cup L''''_2 = L'''_1$ and $L''''_1 \cap L''''_2 = \emptyset$

```

1  $activities \leftarrow dict(key = activity, value = af_{L'''_1}(activity));$ 
2 forall the  $variant \in L'''_1$  do
3   forall the  $activity \in variant$  do
4     if  $activity \notin activities$  then
5        $activities = activities \cup \{activity\};$ 
6        $af_{L'''_1}(activity) \leftarrow vf_{L'''_1}(variant);$ 
7     else
8        $af_{L'''_1}(activity) \leftarrow af_{L'''_1}(activity) + vf_{L'''_1}(variant)$ 
9     end
10  end
11 end
12  $activities \leftarrow \text{sort } activities \text{ by } af_{L'''_1}(activity);$ 
13  $nr\_activities \leftarrow |activities|;$ 
14  $indices \leftarrow \bigcup_{i=0}^p \{n \in \mathbb{N}_0 | n \in [round(min_i \times nr\_activities), round(max_i \times nr\_activities)]\};$ 
15  $filtered\_activities \leftarrow \bigcup_{i \in indices} activities_i;$ 
16  $L''''_1, L''''_2 \leftarrow [];$ 
17 forall the  $trace \in L'''_1$  do
18    $new\_trace \leftarrow [];$ 
19    $not\_new\_trace \leftarrow [];$ 
20   forall the  $activity \in trace$  do
21     if  $activity \in filtered\_activities$  then
22        $new\_trace = new\_trace \cup \{activity\};$ 
23     else
24        $not\_new\_trace = not\_new\_trace \cup \{activity\};$ 
25     end
26   end
27   if  $new\_trace \neq \emptyset$  then
28      $L''''_1 \leftarrow L''''_1 \cup \{new\_trace\};$ 
29   end
30   if  $not\_new\_trace \neq \emptyset$  then
31      $L''''_2 \leftarrow L''''_2 \cup \{not\_new\_trace\};$ 
32   end
33 end

```

---

to L''''_1 only in case it is not empty, i.e., it contains at least one of the activities that should remain. Same goes for L''''_2 .

The filters presented so far rely on the essential attributes of the events in an event log: case identifier, activity label and timestamp. However, the events in real-life event logs can often contain attributes and it might be necessary to filter on them as well. Algorithm 3.3 presents the configurable filter that allows to filter on any additional attribute. Apart from the ranges, it also takes the name of the attribute as input. The allowed ranges, as discussed above, vary depending on whether the attribute is numeric (lines 12–18 and 44–60) or string (lines 20–39 and 62–92). It also distinguishes between filtering on trace attributes (lines 11–41) or event attributes (lines 42–94) using the checks on lines 3–9.

It must be also taken into account that the output sublogs of each filter together build up its input log. However, the overall relation of the initial log and the resulting log is slightly more complex. As advertised in the beginning, our technique splits the initial log L into two logs L_1 and L_2 such that $L_1 \cup L_2 = L$ and $L_1 \cap L_2 = \emptyset$. L_1 is the output L''_1 of

Algorithm 5: Filter on attribute

Input: Event log $L_1^{''''}$. Attribute name *attribute*, Ranges $R_{at} = \{(min_0, max_0), \dots, (min_q, max_q)\}, q \in \mathbb{N}_0$

Result: New event logs $L_1^{''''} \subseteq L_1^{''''}$ and $L_2^{''''} \subseteq L_1^{''''}$ such that $L_1^{''''} \cup L_2^{''''} = L_1^{''''}$ and $L_1^{''''} \cap L_2^{''''} = \emptyset$

```

1 Begin
2    $traces\_with\_attr, not\_traces\_with\_attr, L_1^{''''}, L_2^{''''} \leftarrow [];$ 
3   forall the  $trace \in L_1^{''''}$  do
4     if  $\exists \#_{attribute}(trace)$  then
5        $traces\_with\_attr \leftarrow traces\_with\_attr \cup \{trace\};$ 
6     else
7        $not\_traces\_with\_attr \leftarrow$ 
8          $not\_traces\_with\_attr \cup \{trace\};$ 
9     end
10  end
11  if  $traces\_with\_attr \neq \emptyset$  then
12    if  $type(attribute) \in \{float, int\}$  then
13      forall the  $trace \in traces\_with\_attr$  do
14        if  $\exists (min_n, max_n) \in A. (min_n \leq \#_{attribute}(trace) \leq max_n)$  then
15           $L_1^{''''} \leftarrow L_1^{''''} \cup \{trace\};$ 
16        else
17           $L_2^{''''} \leftarrow L_2^{''''} \cup \{trace\};$ 
18        end
19      end
20    else
21       $attribute\_frequencies \leftarrow dict(key =$ 
22         $attribute\_value, value =$ 
23         $atf_{L_1^{''''}}(attribute, attribute\_value));$ 
24      forall the  $trace \in traces\_with\_attr$  do
25        if  $\#_{attribute}(trace) \notin attribute\_frequencies$ 
26          then
27             $atf_{L_1^{''''}}(attribute, \#_{attribute}(trace)) \leftarrow$ 
28               $1;$ 
29          else
30             $atf_{L_1^{''''}}(attribute, \#_{attribute}(trace)) \leftarrow$ 
31               $atf_{L_1^{''''}}(attribute, \#_{attribute}(trace)) + 1;$ 
32          end
33      end
34       $attribute\_frequencies \leftarrow sort$ 
35         $attribute\_frequencies$  by
36         $atf_{L_1^{''''}}(attribute, attribute\_value);$ 
37       $nr\_values \leftarrow |attribute\_frequencies|;$ 
38       $indices \leftarrow \bigcup_{i=0}^q \{n \in \mathbb{N}_0 | n \in [round(min_i \times$ 
39         $nr\_values), round(max_i \times nr\_values)]\};$ 
40       $filtered\_values \leftarrow \bigcup_{i \in indices} attribute\_frequencies_i;$ 
41      forall the  $trace \in traces\_with\_attr$  do
42        if  $\#_{attribute}(trace) \in filtered\_values$  then
43           $L_1^{''''} \leftarrow L_1^{''''} \cup \{trace\};$ 
44        else
45           $L_2^{''''} \leftarrow L_2^{''''} \cup \{trace\};$ 
46        end
47      end
48    end
49     $L_2^{''''} \leftarrow L_2^{''''} \cup not\_traces\_with\_attr;$ 
50  end

```

---

the last applied filter, meaning $L = L_1^{''''}$ in case the custom attribute filter is applied. L_2 , however, follows a different

Algorithm 5: Filter on attribute (cont.)

```

42 else
43   if  $type(attribute) \in \{int, float\}$  then
44     forall the  $trace \in L_1^{''''}$  do
45        $new\_trace \leftarrow [];$ 
46        $not\_new\_trace \leftarrow [];$ 
47       forall the  $e \in trace$  do
48         if  $\exists \#_{attribute}(e) \wedge \exists (min_n, max_n) \in$ 
49            $A. (min_n \leq \#_{attribute}(e) \leq max_n)$  then
50            $new\_trace = new\_trace \cup \{e\};$ 
51         else
52            $not\_new\_trace =$ 
53              $not\_new\_trace \cup \{e\};$ 
54         end
55       end
56       if  $new\_trace \neq \emptyset$  then
57          $L_1^{''''} \leftarrow L_1^{''''} \cup \{new\_trace\};$ 
58       end
59       if  $not\_new\_trace \neq \emptyset$  then
60          $L_2^{''''} \leftarrow L_2^{''''} \cup \{not\_new\_trace\};$ 
61       end
62     end
63   else
64      $attribute\_frequencies \leftarrow dict(key =$ 
65        $attribute\_value, value =$ 
66        $atf_{L_1^{''''}}(attribute, attribute\_value));$ 
67     forall the  $trace \in L_1^{''''}$  do
68       forall the  $e \in trace$  do
69         if  $\#_{attribute}(e) \notin attribute\_frequencies$ 
70           then
71              $atf_{L_1^{''''}}(attribute, \#_{attribute}(e)) \leftarrow$ 
72                $1;$ 
73           else
74              $atf_{L_1^{''''}}(attribute, \#_{attribute}(e)) \leftarrow$ 
75                $atf_{L_1^{''''}}(attribute, \#_{attribute}(e)) + 1;$ 
76           end
77       end
78     end
79      $attribute\_frequencies \leftarrow sort$ 
80        $attribute\_frequencies$  by
81        $atf_{L_1^{''''}}(attribute, attribute\_value);$ 
82      $nr\_values \leftarrow |attribute\_frequencies|;$ 
83      $indices \leftarrow \bigcup_{i=0}^q \{n \in \mathbb{N}_0 | n \in [round(min_i \times$ 
84        $nr\_values), round(max_i \times nr\_values)]\};$ 
85      $filtered\_values \leftarrow \bigcup_{i \in indices} attribute\_frequencies_i;$ 
86     forall the  $trace \in L_1^{''''}$  do
87        $new\_trace \leftarrow [];$ 
88        $not\_new\_trace \leftarrow [];$ 
89       forall the  $e \in trace$  do
90         if  $\exists \#_{attribute}(e) \wedge \#_{attribute}(e) \in$ 
91            $filtered\_values$  then
92            $new\_trace = new\_trace \cup \{e\};$ 
93         else
94            $not\_new\_trace =$ 
95              $not\_new\_trace \cup \{e\};$ 
96         end
97       end
98       if  $new\_trace \neq \emptyset$  then
99          $L_1^{''''} \leftarrow L_1^{''''} \cup \{new\_trace\};$ 
100      end
101       if  $not\_new\_trace \neq \emptyset$  then
102          $L_2^{''''} \leftarrow L_2^{''''} \cup \{not\_new\_trace\};$ 
103       end
104     end
105   end
106 end

```

---

logic as it comprises deselected behavior of the five filters, thus $L_2 = L'_2 \cup L''_2 \cup L'''_2 \cup L''''_2 \cup L'''''_2$.

4 Evaluation

In this section, we evaluate our prototype. The evaluation is divided as follows. First, we describe the implementation of the prototype. Second, we evaluate the performance of the tool on simulated logs of varying length and trace size. Third, we demonstrate the effectiveness of our technique by applying our technique to simulated data from the running example we provided in Sect. 2.1. Fourth, we show the usefulness of our technique in a real-life log. To conclude, we present the results from a user evaluation of the tool.

4.1 Prototypical implementation

We implemented our technique as a prototype, which we call *InterLog*. We built our prototype using version 2.0.0 of the PM4Py [3] library. It is a library for process mining implemented in the Python programming language. As we were building an interactive tool, we used Django¹ web framework for our implementation. We packed our application into a Docker container and deployed it on our institute's Kubernetes cluster in order to allocate resources to it as needed. Due to this high degree of virtualization and flexible resource allocation, we cannot provide exact hardware specifications; however, we must note that during our evaluation RAM usage did not exceed 2 GB and CPU usage did not exceed 0.5 cores.

Our tool takes an event log in XES format as input. The output comprises two event logs, which are partitions of the input log, both in XES format. In addition, our tool also provides directly follows graphs generated from these logs, in order to support interactivity. The output logs can also be used with any other process mining tool. Apart from mining a resulting log with the tool's built-in Heuristics Miner based on PM4Py, the user can export it and work on it with other tools like ProM, Disco, Celonis, etc. We, however, relied solely on our tool in our evaluation. Our prototype is publicly available as open source software on GitHub².

Figure 3 shows a screenshot of the user interface (UI) of *InterLog*. After the user has uploaded an event log, the UI presents the output organized in three vertical panes. On the left pane, a process model that was mined from the entire event log is shown. On the middle pane, sliders and input boxes are offered to the analyst. From this pane it is possible to set the number of filters along with the desired ranges. Users can add or remove ranges according to their needs. The multi-range filtering offered by the UI are, respectively,

time, variants, performance and custom. Custom filter allows for multi-range filtering on any attribute of the event log. By default this filter is not enabled, showing the selection Empty filter. In the bottom of the middle pane, the user can select the desired model visualization technique and whether to compute the Levenshtein's distance between the two partitions of the event log. It is shown in [20] as a widely used measure for average distance between the traces in a log. We adapt this measure to compare every trace of the selected log L_1 with every trace in the deselected log L_2 . The right pane is further divided into two areas. At the top of this pane, we find an area with a blue background. Here the analyst can observe a model that was mined from the part of the event log that was filtered in. At the bottom, we find an area with a red background. Here the analyst can observe a model that was mined from the part of the event log that was filtered out, which is complementary to the data used in the blue area above.

4.2 Computational performance

To evaluate the computational performance, we use a similar approach to Di Ciccio et al. [12]. Utilizing their event log simulator, we generated several artificial event logs and recorded the time our tool took to generate an output. A prerequisite of using the event log generator is a DECLARE [21] model. As our technique is not concerned with the semantic of the activities present in such event log, we could use any kind of process. Thus, we used a simple article submission process as described by the DECLARE templates in Listing 1.

```
RespondedExistence("Submit abstract",
    "Write new paper")
Response("Submit paper", "Send
    confirmation email")
Succession("Submit paper", "Review
    paper")
Precedence("Review paper", "Accept
    paper")
NotSuccession("Reject paper", "Submit
    paper")
NotCoexistence("Accept paper", "
    Reject paper")
```

Listing 1 Scientific paper evaluation process

We set the parameters as follows. We varied the logs size and maximum trace length. As our filters also perform a vertical slicing of the event log, we decided to keep the minimum trace length at 2. In this way, we account for more realistic logs and can observe if the variability of trace length is correctly handled. We generate logs with maximum trace lengths of, respectively, 500, 1000, 1500, 2500 and 5000 events. These are five points among multiples of 500, similarly to the approach in [12]. As per log sizes, we generated

¹ <https://www.djangoproject.com/>

² <https://github.com/MaxVidgof/interlog>

Fig. 3 UI of InterLog

logs of, respectively, 10, 100, 1000 and 10000 traces. Given that we evaluate three types of filters, this sums up to a total of 20 generated event logs.

We evaluate the performance of our filtering technique based on the following consideration. The devised filtering algorithms fall into three categories: time-bound, trace-bound, event-bound filters. Time-bound filters partition the log into traces in such a way that all the events of the trace fall within a specified time interval. The performance of this operation depends on the values of the timestamp contained in each trace. Trace-bound filters partition the event log based on the frequency of their traces. The performance of this operation depends on the sorting of the traces by frequency. Event-bound filters partition the event log based on the presence of a given frequency of events (e.g., activities). To this end, they perform a sorting of the events, a selection of the events that fall within the specified ranges and a partition of the traces into the ones which contain those events and the ones which do not contain the selected events. The performance of this type of filter depends on both the number of traces and the number of different types of events in the log.

Specifically we map the filtering techniques into categories as follows. Filter on timestamp and filter on performance are time-bound. Variants filter and any custom filter about a trace attribute are trace-bound. Filter activities and any custom filter about an event are event-bound.

With this categorization, we calculate the performance of the InterLog tool on the filters of time, variants and activities, each of them being a representative of the categories time-bound, trace-bound and event-bound, respectively.

Table 3 summarizes the generated event logs and reports the execution time of each filter. The filters were applied independently on the event logs, hence not interfering with one another. The table reports the time in seconds, for each test. Rows Avg and StD summarize, respectively, the average and standard deviation.

A number of observations can be derived from Table 3. First, we compare how the performance changes with respect to the size of the event log. As the number of traces in the log grows exponentially, the computation time also grows exponentially. Thus, the running time of our algorithms increases linearly in the size of the log. Figure 4 depicts this comparison. As the reader can notice, the performance is similar on event logs with maximum trace length of 500 events (Fig. 4a) and with 5000 events (Fig. 4b). Thus, the trace length does not significantly impact the performance growth.

Second, we compare how the performance of each filtering technique changes with respect the growth of trace length. Figure 5 shows such comparison. The x-axis represents the number of traces, while the y-axis represents the computation time in logarithmic scale. We applied logarithmic scale on the y-axis to ease the visualization of the exponential growth of size of the input event logs. Each event log is depicted with a different line color. As we can observe, the growing size of the traces is related to the computation time by a constant value, as shown by the nearly horizontal lines. Thus, we can confirm that (i) the trace length does not appear to have a significant impact on performance and (ii) the computation time appears to grow proportionally to the growth of the event log size.

Table 3 Time (seconds) to partition to event log: trace maximum length versus log size

Trace size	Time filter				Variants filter				Activities filter			
	10 ¹	10 ²	10 ³	10 ⁴	10 ¹	10 ²	10 ³	10 ⁴	10 ¹	10 ²	10 ³	10 ⁴
500	0.0008	0.0021	0.0155	0.8882	0.0006	0.0039	0.0310	1.3945	0.0003	0.0022	0.0202	0.7656
1000	0.0012	0.0012	0.0149	0.9069	0.0010	0.0042	0.0270	1.3163	0.0004	0.0035	0.0161	0.8844
1500	0.0005	0.0021	0.0147	0.8461	0.0005	0.0045	0.0320	1.2385	0.0003	0.0036	0.0223	0.6425
2500	0.0006	0.0012	0.0154	0.8517	0.0004	0.0026	0.0369	1.4405	0.0003	0.0020	0.0211	0.8033
5000	0.0005	0.0011	0.0154	0.9004	0.0007	0.0024	0.0340	1.4022	0.0004	0.0020	0.0216	0.6540
Avg	0.0007	0.0015	0.0152	0.8786	0.0006	0.0035	0.0322	1.3584	0.0003	0.0027	0.0202	0.7500
StD	0.0003	0.0005	0.0004	0.0281	0.0002	0.0010	0.0037	0.0808	0.0001	0.0008	0.0024	0.1024

Fig. 4 Time to partition the event log when maximum trace length grows **(a)** Log size versus time (max trace length = 500)**(b)** Log size versus time (max trace length = 5000) **(a)** time filter performance (secs)**(b)** activity filter performance (secs)**(c)** variants filter performance (secs)**Fig. 5** Performance to partition the event logs (secs) of the three filters

Finally, we also tested whether the number of user-defined ranges has an impact on the performance. We fixed the event log and varied the number of ranges. As the performance on the biggest event logs falls under 2 seconds, we selected an event log of size 10^4 for this test. Trace length was set

between 2 and 5000 events. We varied the number of ranges in the interval $[1, 10]$. Ranges were chosen in such a way that i) they are evenly distributed over their domain of application (i.e., timestamps, variants, activities) and ii) they have about the same size. For instance, when we set the number

Fig. 6 Performance of each filter versus the number of input ranges

of input intervals $I = 2$, then we would create the ranges $R_1 = [0.25, 0.5]$ and $R_2 = [0.75, 1]$. These ranges have both size 2.5 and are evenly distributed. Thus, the timestamp filter will filter only the events that fall between the percentiles 25-50% and 75%-100% of all the timestamps in the event log. Analogously, the variants and activities filter will apply the filtering to, respectively, the same percentiles of the variants and the activities. Table 4 summarizes the results of the performance tests. It presents for each number of ranges, the number of seconds needed by each filter to perform a partition of the log.

To ease the analysis of the impact of the number of ranges on computation time, we plotted the data given in Table 4 as a column chart. We present this chart in Fig. 6. Surprisingly, no significant impact of the number of input ranges was observed on the performance. In fact, the performance is lowest at ranges is 1, 5 and 10. This suggests that the algorithm is mostly affected by the specific content of the chunks of the event log to be partitioned, rather than their number.

4.3 Effectiveness of multi-range filters

In [29], we generated a log of our example process in Fig. 1 using BIMP³. That log contained 1000 cases and was built with the following rules: (i) positive response is received with 80% probability; (ii) negative response is received with 20% probability; (iii) alternative solution exists with 80% probability; and (iv) no alternative solutions exist with 20% probability.

To best show the effectiveness of the time filtering feature of *InterLog*, we added concept drift to the aforementioned event log, but extending it with 1000 new traces. These traces which follow the same model and same rules, but the activity ‘Send apology’ was performed after the activity ‘Propose solution.’ This led us to a log with 2000 cases, where apart from the different variants we also simulate a process drift.

Let us apply our prototype to this new artificial log. First, we can apply the time filter and set the range of the filter $R_t = \{[‘2020-03-05 09:00:00’, ‘2020-05-18 12:00:00’]\}$. If we apply only this filter and leave the ranges for the other two filters unchanged ($R_v = \{[0, 1]\}$, $R_a = \{[0, 1]\}$), we will see the models in Fig. 7. Although the two process models look similar, there is one crucial difference: in Fig. 7a the activity ‘Send apology’ (D) precedes the activity ‘Discuss solution’ (E), while in Fig. 7a the order of these two activities is the opposite. We can observe that our tool has helped to identify a process drift. The event log was successfully partitioned into a sub-log L_1 that contains the traces before the drift and a sub-log L_2 that contains the traces after the drift.

Leaving the time filter as it is (with $R_t = \{[‘2020-03-05 09:00:00’, ‘2020-05-18 12:00:00’]\}$), we can also apply the variants filter on top of it. From the traces before the drift, we may want to look at the most frequent and the least frequent behavior. To do that, we apply Algorithm 3.3 and specify two ranges for the filter: $R_v = \{[0, 0.15], [0.9, 1]\}$. It means we want to keep the 15% least frequent paths as well as 10% most frequent ones. It is very important to interpret these ranges correctly: by saying we take 15% most infrequent paths we do not mean taking 15% of the cases. Instead, we mean here paths that are between the 0th and the 15th percentile in a list of all variants in the input log sorted by their frequency.

We do not want to apply any other filters at this point, thus we specify one range $R_p = \{[0, 1]\}$ for the performance filter, meaning we do not apply additional filters on the trace level, one range $R_a = \{[0, 1]\}$ for the activities filter, meaning we want to keep 100% of activities, and we set the attribute in the last filter to Empty. This, as always, gives us a two logs L_1 and L_2 , that we can analyze based on the models provided by our tool, or download and use with any other process mining tool. Figure 8 shows a model resulting from applying PM4Py’s directly follows graph visualizer on L_1 .

However, we may also want to filter activities at this point. Note that as we already applied the other two filters filter on our log, only the activities present in the selected traces will be available for us to pick from. Let us say, we want to see the least frequent activities as well as the ones of medium frequency but not the most frequent ones. In order to do that, we can set multiple ranges for the activities filter: $R_a = \{[0, 0.1], [0.1, 0.3], [0.4, 0.6]\}$. You can also see that the range boundaries are allowed to be the same but an overlap between ranges is not allowed.

Figure 9 shows the resulting model. As we can see, it only includes the activities that are in the specified range: 40% least frequent activities and some activities with medium frequency. However, the new model does not contain the most frequent activities as they are outside of the specified range. This allows the user to concentrate on the less frequent and presumably more interesting activities.

³ <http://bimp.cs.ut.ee>

Fig. 7 Models from the artificial log generated from the process in Fig. 1 with time range $R_t = \{[2020-03-05\ 09:00:00, 2020-05-18\ 12:00:00]\}$ produced by Heuristics Miner augmented with labels from Table 1 for improved readability

Fig. 8 Directly follows graph from the artificial log in Table 1 with time range $R_t = \{[2020-03-05\ 09:00:00, 2020-05-18\ 12:00:00]\}$ and variants ranges $R_v = \{[0, 0.15], [0.9, 1]\}$

Table 4 Computation time for each filter type versus the number of input ranges

Filter type	Number of input ranges									
	1	2	3	4	5	6	7	8	9	10
Time	0.9482	0.8207	0.9523	1.0915	1.3383	1.0783	1.0745	1.3384	1.4596	1.7746
Variants	1.7818	0.7005	0.7067	1.2922	2.3690	0.6095	0.5052	1.1787	1.4783	1.9827
Activity	1.2245	0.3547	0.5436	0.6255	1.5307	0.3387	0.4360	0.8054	0.9886	1.6442

Fig. 9 Directly follows graph from the artificial log in Table 1 with time range $R_t = \{[2020-03-05\ 09:00:00, 2020-05-18\ 12:00:00]\}$, variants ranges $R_v = \{[0, 0.15], [0.9, 1]\}$ and activities ranges $R_a = \{[0, 0.1], [0.1, 0.3], [0.4, 0.6]\}$

To sum up, we showed that *InterLog* is able to partition the input into highly customized chunks. Moreover, our technique is preserving important information in two ways. First, it does not penalize infrequent behavior. Second, it provides the user with both the filtered-in and the filtered-out behavior. Moreover, all the event log chunks generated by our technique can be used as input to any process mining technique.

4.4 Applicability on real-life logs

Next, we evaluated our technique on two real-life event logs. First, we used the event log concerning a ticketing management process belonging to the help desk of an Italian software company. [22] This log contains more than 4500 traces and 21000 events covering the time frame between January 13, 2010, and January 3, 2014.

This log includes 226 variants, of which 136 are represented by only one trace each. The most frequent variant,

however, is represented by 2366 traces, which is almost 52% of all traces. This hints that the process is not as streamlined as the simulated ones, however, provides little help in investigating the differences between the frequent and the infrequent variants, let alone the possible reasons for these differences.

What is more useful for this event log is the timestamp filter. By simply cutting the event log in half with the timestamp filter, one may see that the process was executed differently in the first half of the full time span than in the second. To be more precise, one can experiment with the exact range (or ranges) for the time filter—luckily, the tool is interactive enough to let the user modify the ranges and get almost immediate feedback—and eventually land at the following range: $R_t = \{[2010-01-13\ 08:40:25, 2011-07-25\ 08:00:00]\}$. Figure 10 shows the result of splitting this log on the specified timestamp and applying Heuristics Miner to the resulting sub-logs.

With the two logs at hand, the analyst can compare the two complementary logs and notice some differences. For instance, traces in L_2 include new activity ‘Require update’ that was seen in the process before. More notably, in L_2 it is possible to resolve the ticket right after assigning seriousness, while in L_1 there always had to be activity ‘Take in charge ticket’ in between. This could signal a process drift taking place at that moment in time, which might have been caused by a ticketing system update. This example shows that, with the output provided by our tool, the analyst can detect possible process drifts and find questions for deeper investigation.

The second real-life event log we used in our evaluation is the sepsis cases log [19]. This is another publicly available log containing more than 1000 traces and 15000 events, each trace corresponding to a pathway through the hospital.

By exploring the log, we can find out that there are 846 different variants, the most frequent of which includes only 35 cases that corresponds to slightly more than 3% of all traces in the log. There are also 784 variants having only a single conforming trace in the log. This means that the term frequent variant is not applicable to this log. Thus, it makes little sense to apply the variants filter on the log so we can set the range of the variants filter to $[0,1]$.

The event timestamps in the log are randomized for privacy reasons, thus applying the time filter would not be as

Fig. 10 Models from the real-life help desk log from [22] with time range $R_t = \{[2010-01-13\ 08:40:25, 2011-07-25\ 08:00:00]\}$ produced by Heuristics Miner

helpful as in the previous example. Indeed, even if different time partitions in this log exhibit different behavior, it still provides no insights about the underlying process since the traces in these partitions of the public log may in fact be distributed differently in the underlying non-randomized event log. Thus, for this evaluation, the time filter will also remain untouched.

What is really of interest to us is the activities filter. While the filters of the traditional process mining tools only allow to keep the most frequent activities, our filter gives us more opportunities. For instance, we can decide to take a deeper look only into the least frequent activities. For this, we would set the activities filter to a range of $[0, 0.25]$. But we can also add additional ranges to these filter. Let us say, apart from the least frequent activities we are also interested in the one activity lying at the 65th percentile of frequency. This is also possible, for this we just set the second range to $[0.65, 0.65]$.

The resulting directly follows graph can be seen in Fig. 11. Only see the activities that are in the specified range of frequency can be seen in this graph, and such a graph cannot be achieved by any other process mining tool. It must be noted that Fig. 11 only shows the model produced from the log L_1 and focuses solely on the infrequent and moderately frequent activities.

This evaluation shows that InterLog can be effectively used to analyze real-life event logs. Although due to some structural properties of their logs or privacy issues, not all of the InterLog's functions are applicable to all of the logs, it can still provide helpful features for filtering, partitioning and analyzing various event logs from real-life systems.

4.5 User evaluation

In this section, we report how prospective users assess the *InterLog* technique. To this end, we conducted a user study using a survey design in order for capturing both qualitative and quantitative feedback on our technique. The survey consisted out of three parts. In the first part of the study, participants were asked to perform five tasks using InterLog technique. For the purpose of these tasks, we built in sepsis event log in to the tool and asked participants to filter for certain behavior frequent and infrequent behavior, as well as to calculate Levenshtein's distance. The second part of the study was based on the established technology acceptance model [11,28]. This model is used to evaluate two major factors of technology adoption: perceived usefulness and ease of use. In the context of our research, perceived usefulness and ease of use are important antecedents for the intent of process analysts to utilize the InterLog technique in their

Fig. 11 Directly follows graph produced from the real-life sepsis cases log from [19] with activities ranges
 $R_a = \{[0, 0.25], [0.65, 0.65]\}$

daily work. To put it simple, if the technique is judged to be easy to use and it provides useful insights, then it is likely to be adopted. To collect the data on perceived usefulness and ease of use, we opted for using the already established psychometric measurement instrument with 6 and 5 items per construct, respectively [11]. The items were evaluated using 7-point Likert scales, from 1 being ‘I fully disagree’ to 7 being ‘I fully agree.’ The third part of the study consisted of two open-ended questions that were aimed at collecting feedback on aspects of the technique that participants liked, and on aspects that they would like to see improved.

Since previous research showed that there are no significant differences between the cognitive processes of IT professionals and students [7], we decided to recruit master students who previously attended courses related to process mining as participants in our study. The total number of participants was 14. However, one submission was invalid and thus had to be removed from the dataset. This resulted in the final number of 13 participants.

The results of the tasks performed using InterLog show that participants answered roughly half of questions correctly, with the exception of the last task where none of the participants gave a correct answer. We believe that the reason for this is that participants did not reset the values of their respective filters after finishing with the last task. We plan to rectify this in the future and implement more controls, so these situations do not occur again.

Overall, these results suggest that participants responded positively to InterLog technique. The results of the technology acceptance assessment, shown in Fig. 12, suggest that participants rated InterLog highly both in terms of perceived usefulness and ease of use. The average (dashed line) score for perceived usefulness is 5.81 and 5.75 for ease of use, respectively. The median (solid line) scores for perceived usefulness and ease of use are 6.33 and 6.4, respectively. This indicates that the participants consider the tool to be both useful and easy to use for their purposes.

Fig. 12 Boxplots of perceived usefulness and perceived ease of use according to user evaluation study. Solid line is the median, and dashed line is the mean

The qualitative feedback provided insights into what the participants liked best and what they suggest to be improved. Regarding aspects they liked, participants referred to its wide range of filters and visualizations that can be combined to explore the log I like the different types of filters and especially the different types of visualizations. Also the possibility to see the event log at the beginning vs the one after filtering is good.; clear graphical representations;. Also appreciated was the ease of use I find the filters are clearly presented. It is easy to use, also the bars that show how much of the filter is taken in is good.; It is simple to use given the ease of setting up the parameters. It is useful thanks to do the abilities it provides. Lastly, it is self-explanatory (thanks for the explanation signs ‘!’) and thus easy to learn.

When it comes to the aspects that were suggested to be improved, most of the comments referred to the design of the user interface, such as colors, positions of the graphs and filters, while another large part of the feedback was the lack of more detailed user guide. One participant states that ‘The design is user-friendly but I think it could be slightly more elegant.’ Another one wrote: The representations, how-

ever, could be presented with more colors if necessary. The lines are sometimes very messy and overlap each other. Also both filters should be explained whether now from 0-0.3 the ‘fastest’ cases are or at 0.7-1. That was not clear to me., while others noted A better headline for the output traces (description) would be useful—which is what? and Use a friendlier appearance and filters are not well explained. One participant explicitly addressed the need for the user guide I would kindly suggest the implementation of a user handbook, such that everyone can get used to InterLog easily.

5 Discussion of requirements

In Sect. 2.3, we described a set of five requirements for designing an interactive log-delta analysis technique. Based on the evaluation, we summarize the contribution of our InterLog technique as follows:

RQ1 (Select time range) The InterLog technique allows the user to select time ranges and then splits the input log based on these ranges. The traces, all events of which take place within the user-defined time ranges, are retained in L_1 , while the remaining traces form L_2 . The InterLog technique also allows to select performance ranges, so that the traces having their throughput time within the relative ranges are kept in L_1 , while the others build up L_2 .

These filters are extremely useful for manually detecting process drift as well as for analyzing the differences between the process before and after the drift. In this way, they can complement automated drift detection techniques such as [32].

RQ2 (Select traces) The InterLog technique allows the user to select variants based on their frequency. The user can provide frequency ranges, and variants whose frequency fits within these ranges are retained in L_1 , while the remaining variants compose L_2 . In addition, the InterLog technique allows to split the event log on any other user-selected trace attribute, so that the traces having the attribute within the user-selected frequency (string attributes) or value (numeric attributes) ranges remain in L_1 , while the other traces form L_2 .

Variants selection is an essential feature for log-delta analysis. Indeed, in order to compare different chunks of a process’s behavior one has to split the process into these chunks.

RQ3 (Select activities) The InterLog technique allows the user to select activities, also based on their frequency, by providing frequency ranges. The traces in L_1 are filtered in such a way that they include only the activities within the user-specified ranges, while the remaining activities from each trace build up a complementary trace that is

appended to L_2 . In addition, InterLog offers frequency-based filtering on other string event attributes as well as value-based filtering for the numeric event attributes in a similar way as described above.

Activities filtering is useful in two regards. First, it is important for simple event log filtering, as it allows to reduce the number of activities, thus reducing the complexity of the resulting model, which, in turn, tends to make such model easier to comprehend. Second, activities filtering also plays an important role for log-delta analysis. For instance, if the frequent activities are retained in one log partition and the infrequent ones—in the other partition, the analyst can investigate whether the patterns that occur in frequent activities also apply to the infrequent ones.

RQ4 (Multi-range filtering) The InterLog technique allows the user to select multiple non-overlapping ranges for each of the three filters. Thus, our technique provides means for fine-grained selection of the elements in the resulting sub-logs.

The ability to select multiple ranges significantly extends the functionality of each of the three filters in RQ1-3. For instance, the time filter extended with multi-range functionality is able to select different time spans within the process, including not only simple options like splitting the process in half or selecting the beginning or the end of the event log, but also the means to select periodic time spans, which would allow observing periodical changes in the behavior.

The variants filter extended with multi-range feature allows to retain all the necessary variants in L_1 , regardless of whether they are equally frequent or belong to different frequency ranges, as these ranges can be combined. For event log filtering, it brings a wide range of opportunities, such as creating one process model that is simplified though still providing not only the most frequent variant but also variants of different frequencies. These opportunities have already been described in [29]. This multi-range functionality, however, is also of great importance for log-delta analysis. Indeed, with the multi-range split, the analyst is able to compare not only single pairs of variants or frequent variants against the infrequent ones but also all user-specified variants on the one hand versus ‘all the rest’ on the other hand.

The selection of multiple ranges for activities filter provides benefits similar to the ones of the multi-range variants filtering for the filtering of event logs. Indeed, the ability of InterLog to retain activities of multiple frequency ranges allows to further simplify the models resulting from filtering without leaving out important behavior even if it is infrequent. In addition, multi-range functionality extends the activities filter in such a way that effectively any partitioning of activities is possible.

This means the analyst can not only search and compare patterns in frequent and infrequent activities but pick any subset of activities that are related in some way and compare it with the remaining activities in the event log.

RQ5 (Interactive partitioning) The InterLog technique, implemented as a web application, provides a rich user interface, allowing to dynamically add, remove and modify ranges for each of the filters. After the filters are applied, InterLog does not only output the partitioned event logs but provides their visualization with state-of-the-art process mining techniques. Finally, after receiving such output, the user can further adapt his selected ranges until a desired partitioning is reached.

Interactivity added on top of InterLog's functionality boosts its utility even further. With InterLog, the analyst receives feedback on his input in near real time, which generally increases the analyst's productivity by allowing him to spend most of the time on data analysis and not on formatting input or waiting for the output of the tool. Execution time of InterLog's functions lies within 1 second range and grows linearly with input log size while staying almost constant with increasing trace length and number of ranges, which means the interactivity of the tool will not suffer as the size of the input increases. In addition, InterLog allows to quickly adapt the input and rerun event log filtering or partitioning, which allows the analyst to quickly react to undesired results or errors. This interactivity, together with an intuitive and user-friendly interface, make InterLog a very easy-to-use tool that does not require extensive training, which leads not only to even higher productivity but also to increased employee satisfaction.

The evaluation of the prototype shows that InterLog successfully meets all the requirements for an interactive log-delta analysis technique defined in Sect. 2.3. The contributions of our paper therefore include not only the possibility of combining multi-range filters to inspect typical and atypical behavior event log, but also a to do this in an interactive manner. Therefore, to the best of our knowledge, InterLog is the first tool that provides users with this functionality, and supports all of above-defined requirements.

6 Conclusion

A key analysis task for process analysts is to understand the distinctive features of different variants of the process and their impact on process performance. Techniques for log-delta analysis (or variant analysis) mostly build on automatic techniques, but provide limited support for interactively exploring the dividing line between typical and atypical behavior. In this paper, we addressed this research

gap by developing and evaluating an interactive technique for log-delta analysis, which we call *InterLog*. By interactively partitioning the log, the analyst can manually separate the typical behavior from the atypical. We implemented *InterLog* as a prototype and demonstrated its application for a real-world event log. Furthermore, we evaluated it in a preliminary design study with process mining experts for usefulness and ease of use.

There are different directions of future research to further extend *InterLog*. First, we plan to integrate operations of automatic log-delta analysis in a way that the analyst can use them deliberately during an interactive exploration of the variants. Second, we see opportunities by exploring different ways of visualizing the intermediate results of the log-delta analysis. At this stage, we utilized directly follows graphs. For certain types of analysis, also dotted charts or Petri nets might be useful. Finally, we plan to further refine our prototypical implementation based on the feedback we obtained from the user study.

Funding Open access funding provided by Vienna University of Economics and Business (WU).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

A Appendix

A.1 Tasks performed with InterLog

First part of the survey consists of five tasks that need to be performed on the Sepsis log within the tool.

1. Given the input event log, please perform the following actions on InterLog:
  - Filter for all the events that happened within the first three months of year 2014
  - Focus on the 10% least frequent process variants

How many traces have remained after filtering in the selected log?

How many traces have remained after filtering in the deselected log?

2. Given the input event log, please perform the following actions on InterLog:

- Filter for all the events that happened within the first three months of year 2014
- Focus on the 20% least frequent process variants
- Focus on the 50% fastest performing cases

How many traces have remained after filtering in the selected log?

How many traces have remained after filtering in the deselected log?

3. Given the input event log, please perform the following actions on InterLog:

- filter for all the events that happened within the first three months of year 2014 and all the events that happened within the first three months of year 2015
- Focus on the 30% least frequent process variants
- Focus on the 40% fastest performing cases.
- Furthermore, please calculate the Levenshtein distance between the filtered logs.

How many traces have remained after filtering in the selected log?

How many traces have remained after filtering in the deselected log?

What is the value of Levenshtein's distance?

4. Given the input event log, please perform the following actions on InterLog:

- filter for all the events that happened within the first three months of year 2014 and all the events that happened within the first three months of year 2015
- Focus on the 60% least frequent variants
- Focus on the 70% fastest performing cases
- Focus on the resources with at least 40 years of age

How many traces have remained after filtering in the selected log?

How many traces have remained after filtering in the deselected log?

5. Given the input event log, please perform the following actions on InterLog:

- Filter for all the events that happened from the start of year 2014 until the end of May 2015
- Focus on variants whose frequency is between 5 and 30% (boundaries included) and those whose frequency is between 40 and 95% (boundaries included)
- Focus on the 40% slowest performing cases
- Focus on those activities whose frequency is between 10 and 30% and those whose frequency is between 40 and 90%, boundaries included
- Calculate Levenshtein's distance

How many traces have remained after filtering in the selected log?

How many traces have remained after filtering in the deselected log?

What is the value of Levenshtein's distance?

A.2 Questions based on technology acceptance model

In the second section of the survey, we included tasks based on the technology acceptance model which aimed to capture quantitative data about participants' perceived usefulness and ease of use of the InterLog tool. The questions were answered using a 7-point Likert scale ranging from 1 ('I fully disagree') to 7 ('I fully agree'). Below you can find the list of questions.

A.2.1 Perceived usefulness

1. Using a InterLog to analyze the event log would enable me to accomplish my analysis tasks more quickly.
2. Using the InterLog would improve my performance of analyzing the event log.
3. Using the InterLog would increase my analysis productivity.
4. Using the InterLog would enhance my effectiveness on the analysis job.
5. Using the InterLog would make it easier to do my analysis job.
6. I would find the InterLog useful for my analysis job.

A.2.2 Ease of Use

1. Learning to operate the InterLog would be easy for me.
2. I would find it easy to get the InterLog to do what I want it to do.
3. My interaction with the InterLog would be clear and understandable.
4. It would be easy for me to become skillful at using the InterLog.
5. I would find the InterLog easy to use.

A.3 Qualitative feedback questions

In the third section of the survey, we included two open-ended questions that was aimed at collecting feedback on the tool functionalities that participants deemed as good, and on the things that they would like to see improved.

1. What did you like about the InterLog (write 3-4 full sentences)
2. What would you wish about improving the InterLog (write 3-4 full sentences)

References

- Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: review and benchmark. *IEEE Trans. Knowl. Data Eng.* **31**(4), 686–705 (2019)
- Ballambettu, N.P., Suresh, M.A., Bose, R.J.C.: Analyzing process variants to understand differences in key performance indices. In: *International Conference on Advanced Information Systems Engineering*. pp. 298–313. Springer (2017)
- Berti, A., van Zelst, S.J., van der Aalst, W.M.P.: Process mining for python (pm4py): Bridging the gap between process- and data science. *CoRR* [arXiv:1905.06169](https://arxiv.org/abs/1905.06169) (2019)
- Bolt, A., de Leoni, M., van der Aalst, W.M.: A visual approach to spot statistically-significant differences in event logs based on process metrics. In: *International Conference on Advanced Information Systems Engineering*. pp. 151–166. Springer (2016)
- Bolt, A., de Leoni, M., van der Aalst, W.M.: Process variant comparison: using event logs to detect differences in behavior and business rules. *Inf. Syst.* **74**, 53–66 (2018)
- Boltenhagen, M., Chatain, T., Carmona, J.: Model-based trace variant analysis of event logs. *Information Systems* p. 101675 (2020). <https://doi.org/10.1016/j.is.2020.101675>, <http://www.sciencedirect.com/science/article/pii/S0306437920301307>
- Compeau, D., Marcolin, B., Kelley, H., Higgins, C.: Research commentary-generalizability of information systems research using student subjects-a reflection on our practices and recommendations for future research. *Inf. Syst. Res.* **23**(4), 1093–1109 (2012)
- Conforti, R., La Rosa, M., ter Hofstede, A.H.: Filtering out infrequent behavior from business process event logs. *IEEE Trans. Knowl. Data Eng.* **29**(2), 300–314 (2016)
- Conforti, R., Rosa, M.L., ter Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. *IEEE Trans. Knowl. Data Eng.* **29**(2), 300–314 (2017)
- Cordes, C., Vogelgesang, T., Appelrath, H.J.: A generic approach for calculating and visualizing differences between process models in multidimensional process mining. In: *International Conference on Business Process Management*. pp. 383–394. Springer (2014)
- Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly* pp. 319–340 (1989)
- Di Ciccio, C., Bernardi, M.L., Cimitile, M., Maggi, F.M.: Generating event logs through the simulation of declare models. In: *EOMAS@CAiSE. Lecture Notes in Business Information Processing*, vol. 231, pp. 20–36. Springer (2015)
- Du, F., Shneiderman, B., Plaisant, C., Malik, S., Perer, A.: Coping with volume and variety in temporal event sequences: strategies for sharpening analytic focus. *IEEE Trans. V. Computer Gr.* **23**(6), 1636–1649 (2016)
- Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: *Fundamentals of business process management*, 2nd edn. Springer, New York (2018)
- Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In: *BPM. Lecture Notes in Computer Science*, vol. 4714, pp. 328–343. Springer (2007)
- Lakshmanan, G.T., Rozsnyai, S., Wang, F.: Investigating clinical care pathways correlated with outcomes. In: *Business process management*, pp. 323–338. Springer (2013)
- Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: *Petri Nets. Lecture Notes in Computer Science*, vol. 7927, pp. 311–329. Springer (2013)
- Lo, D., Cheng, H., Han, J., Khoo, S.C., Sun, C.: Classification of software behaviors for failure detection: a discriminative pattern mining approach. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. pp. 557–566 (2009)
- Mannhardt, F.: Eindhoven University of Technology. Dataset. Sepsis Cases - Event Log. (2016). <https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460>
- Pentland, B.T.: Conceptualizing and measuring variety in the execution of organizational work processes. *Manag. Sci.* **49**(7), 857–870 (2003)
- Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, 15–19 October 2007, Annapolis, Maryland, USA. pp. 287–300. IEEE Computer Society (2007). <https://doi.org/10.1109/EDOC.2007.14>
- Polato, M.: Eindhoven University of Technology. Dataset. Dataset belonging to the help desk log of an Italian Company. (2017). <https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb>
- Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace clustering in process mining. In: *Business Process Management Workshops. Lecture Notes in Business Information Processing*, vol. 17, pp. 109–120. Springer (2008)
- Taymouri, F., La Rosa, M., Carmona, J.: Business process variant analysis based on mutual fingerprints of event logs. In: *International Conference on Advanced Information Systems Engineering*. pp. 299–318. Springer (2020)
- Taymouri, F., La Rosa, M., Dumas, M., Maggi, F.M.: Business process variant analysis: survey and classification. *Knowl-Based Syst.* **35**, 106557 (2020)
- van Beest, N.R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Log delta analysis: Interpretable differencing of business process event logs. In: *International Conference on Business Process Management*. pp. 386–405. Springer (2016)
- van der Aalst, W.M.P.: *Process mining - data science in action*, 2nd edn. Springer, New York (2016)
- Venkatesh, V., Morris, M.G., Davis, G.B., Davis, F.D.: User acceptance of information technology: Toward a unified view. *MIS quarterly* pp. 425–478 (2003)
- Vidgof, M., Djurica, D., Bala, S., Mendling, J.: Cherry-picking from spaghetti: Multi-range filtering of event logs. In: *Enterprise, Business-Process and Information Systems Modeling*, pp. 135–149. Springer (2020)
- Weerd, J.D., van den Broucke, S.K.L.M., Vanthienen, J., Baesens, B.: Active trace clustering for improved process discovery. *IEEE Trans. Knowl. Data Eng.* **25**(12), 2708–2720 (2013)
- Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM). In: *CIDM*. pp. 310–317. IEEE (2011)
- Yeshchenko, A., Di Ciccio, C., Mendling, J., Polyvyanyy, A.: Visual drift detection for sequence data analysis of business processes. *IEEE Transactions on Visualization and Computer Graphics* pp. 1 (2021). <https://doi.org/10.1109/TVCG.2021.3050071>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Maxim Vidgof is a Teaching and Research Associate with the Institute for Data, Process and Knowledge Management at Vienna University of Economics and Business (WU Wien), Austria. Maxim received his bachelors degree in Business, Economics and Social Sciences with major in Information Systems from WU, a bachelors degree in Economics with honors from Don State University of Technology (Rostov-on-Don, Russia), and a masters degree in Information Systems

with honors from WU. His research interests are process mining and robotic process automation. Maxim is a member of the Austrian Society for Process Management.

Djordje Djurica is a Teaching and Research Associate at Vienna University of Economics and Business (WU Wien). He received his bachelors degree in Economics from the University of Novi Sad and a masters degree in Information Systems from WU Wien. Djordje's research is focused on conceptual modeling. In particular, he is interested in human interaction with conceptual models (i.e., business process and decision models) and how such models can be improved by using

cognitive effective techniques.

Saimir Bala is a PhD candidate in Economics and Social Sciences at the Institute for Data, Process and Knowledge Management. He joined the Business Process Management (BPM) team in 2014 and has been involved in various projects, including the SHAPE FFG Project. Saimir's PhD thesis is investigating the ways to bridge the fields of process mining and software engineering, in order to provide data-driven analysis techniques for the process of software development.

His research interests include various topics on BPM area, software engineering, process mining, and text mining. Saimir holds a BSc and an MSc in Distributed Systems from Sapienza University of Rome. In his masters thesis, he developed a framework for an effective implementation of a fault-tolerant web-service orchestrator, in the context of the GreenerBuildings EU project.

Jan Mendling is the Einstein-Professor of Process Science with the Department of Computer Science at Humboldt-Universität zu Berlin, Germany. His research interests include various topics in the area of business process management and information systems. He has published more than 450 research papers and articles, among others in MIS Quarterly, ACM Transactions on Software Engineering and Methodology, IEEE Transactions on Software Engineering, Journal of the Association

of Information Systems and Decision Support Systems. He is a department editor for Business and Information Systems Engineering, member of the board of the Austrian Society for Process Management, one of the founders of the Berlin BPM Community of Practice, and a member of the IEEE Task Force on Process Mining. He is co-author of the textbooks Fundamentals of Business Process Management, Second Edition, and Wirtschaftsinformatik, 12th Edition.