Repairing Outlier Behavior in Event Logs using Contextual Behavior

Mohammadreza Fani Sani[*,a], Sebastiaan J. van Zelst[a,b], Wil M. P. van der Aalst[a,b]

[a] Process and Data Science Chair, RWTH Aachen University, 52056 Aachen, Germany
[b] Fraunhofer Institute for Applied Information Technology (FIT), Konrad-Adenauer-Strasse, 53754 Sankt Augustin, Germany

Abstract. *It is common in practice, e. g., due to logging errors in information systems or the presence of exeptional process behavior, to have outlier behavior in real event data. Such behavior often leads to incomprehensible, complex, and inaccurate analysis results and makes correct and/or important behavior undetectable. In this paper, we propose a novel data preprocessing method, that detects and subsequently repairs outlier behavior in event data. We propose a probabilistic method that detects outlier behavior based on the occurrence probability of a sequence of activities among its surronding contextual behavior. We replace the outlier behavior with more probable behavior among that behavioral context. Our approach allows to remove outlier behavior, which enables us to obtain a more global view of the process. The proposed method has been implemented in both the* PROM- *and the* RAPIDPROM *frameworks. Using these implementations, we conducted several experiments that show that most types of outlier behavior in event data are detectable and repairable via the proposed method. The evaluation clearly demonstrates that we are able to improve process discovery results by repairing event logs upfront. Results show that using the proposed method we obtain more understandable process models with higher accuracy.*

Keywords. Process Mining • Data Cleansing • Log Repair • Event Log Preprocessing • Conditional Probability • Outlier Detection

Communicated by Milena Stróżyna. Received 2018-11-07. Accepted on 2019-06-30.

1 Introduction

Process Mining bridges the gap between traditional data mining and business process management analysis (Aalst 2011). The main subfields of process mining are 1) *process discovery*, i.e, finding a descriptive model of the underlying process, 2) *conformance checking*, i.e, monitoring and inspecting whether the execution of the process in reality conforms to the corresponding designed (or discovered) reference process model, and 3) *enhancement*, i.e, the improvement of a process model, based on the related event data (Aalst 2016). With process mining we discover knowledge from event data, also referred to as *event logs*, readily available in most modern information systems. In all these subfields, event logs are used as a starting point. An event log is a collection of events extracted in the context of a process that indicates which activity has happened at a specific time.

Many process mining techniques, in any of these subfields, work under the assumption that the behavior related to the execution of the underlying process is stored *correctly* within the event log. Moreover, completeness of the recorded behavior, i. e., each instance of the process, as stored in the event log, is already finished, is assumed as well. However, real event data often contains inaccurate or corrupt behavior that is/should not be part of the process (Hernández and Stolfo 1998). This is common in practice, e. g., due to logging errors, human mistakes and the inaccuracy of logging

* Corresponding author.
E-mail. fanisani@pads.rwth-aachen.de

tools. Furthermore, an event log often contains data related to infrequent behavior, i. e., behavior that is rather rare due to the handling of exceptional cases. Here, we use the term *outlier* to consider both noise and infrequent behavior. The presence of such behavior makes most process mining algorithms return inexplicable, incomprehensible or even inaccurate results. To reduce these negative effects, process mining projects often comprise of a preprocessing phase in which one tries to detect and remove traces that contain such undesired behavior (Andrews et al. 2018). This cleaning phase is usually performed manually and it is therefore rather time consuming and costly.

Despite the negative impacts of the presence of outlier behavior, little research has been done towards automated data cleansing techniques that help to improve process mining results. Recently, research has been performed aiming at *filtering* out traces/behavior that contain outlier behavior from an event log (Conforti et al. 2017; Fani Sani et al. 2017). Even though both techniques show improvements in process discovery algorithm results, only a little fragment of outlier behavior within a trace of event data leads to ignoring the trace as a whole. This problem potentially leads to a distortion of the general distribution of common behavior of the process, yielding potentially wrong process mining results.

Therefore, we propose a general purpose *repair method* that, given an event log that potentially contains outlier behavior, detects and modifies such behavior in order to obtain a more reliable input for all possible process mining algorithms. In particular, we use a probabilistic method to identify outlier behavior according to the behavioral *context* of a process, i. e., fragments of activity sequences that occur before and after the potential outlier behavior. After detecting a fragment of outlier behavior, it is replaced with behavior that is more probable to occur given the context in which the outlier behavior occurs.

Using the PROM-based (Aalst et al. 2009) extension of RAPIDMINER, i. e., RAPIDPROM (Aalst et al. 2017), we study the usefulness of our approach, using both synthetic and real event data. The experimental results show that the proposed approach adequately detects and repairs outlier behavior, and as a consequence, increases the overall quality of results of several process discovery algorithms. Additionally, we show that our method is able to improve process discovery results compared to a state-of-the-art existing event log filtering technique (Fani Sani et al. 2017).

This paper extends the work Fani Sani et al. (2018b). Here, we formally define the proposed method and explain it with more details. The proposed method is also applied on many new available real event logs with state-of-the-art process discovery algorithms, i. e., the Inductive Miner, the Split Miner and the ILP Miner. Also, in this paper we used complexity metrics to evaluate the understandability of discovered process models.

The remainder of this paper is structured as follows. Sect. 2 motivates the need for data cleansing and repair methods in the context of process mining. In Sect. 3, we discuss related work, and Sect. 4 defines some preliminary notations. We present our proposed outlier repair method in Sect. 5. Evaluation details and corresponding results are given in Sect. 6. Finally, Sect. 7 concludes the paper and presents directions for future work.

2 Motivation

Real event data often contains outlier behavior. The presence of such behavior causes many problems for process mining algorithms. In particular, most process discovery algorithms incorporate all behavior in event logs as much as possible. As a result, most outlier behavior is incorporated as well, which in general decreases the overall quality of the discovered process models (e. g., by generating flower models), and moreover, increases their complexity. Therefore, it is essential to accurately preprocess event data. In fact, according to the process mining manifesto (Aalst et al. 2011), cleaning event logs is one of the main challenges in the *process mining* field. The statement "garbage in, garbage out", i. e., referring to the fact that low quality data leads to low final quality knowledge (Ribeiro and Zárate 2016) also

applies to the field of process mining. Therefore, accurate preprocessing steps are critical for being able to obtain event data which we consider to be correct and useful for process mining methods.

A naive approach to solve data quality related issues is to remove traces/behavior that seem to describe outlier behavior (Conforti et al. 2017; Fani Sani et al. 2017). In heavy presence of concurrency, these filtering methods remove most of the traces in the event log, i. e., often a large part of the traces is unique. As a result, these approaches, by removing a large part of the traces, potentially jeopardize the statistical distribution of the normal accurate behavior.

For many businesses, all process instances in an event log are valuable and ignoring them is potentially harmful for the trustworthiness of the analyses performed on the basis of the data. For example, in patient treatment in a hospital, recorded over several years, it is undesirable to remove all process related records of a patient just because there exists a small portion of wrongly logged behavior. Also, in some cases, mixed granularities of time-stamps in the logging, e. g., both on a day- and minute level, does not always allow us to obtain a correct ordering of activities. Moreover, because of various reasons such as database errors, one activity may not be recorded in the event log. Because of data gathering policies, it is also possible that we just keep activities that were executed in a specific period of time. This causes to have many incomplete and abrupt process instances.

In such scenarios, after detecting outliers, it is more desirable to repair such behavior instead of removing it. Note that, if there is no noise in an event log, by repairing infrequent behavior and removing too detailed patterns, we may alter correct infrequent behavior into more frequent behavior which allows us to discover more general views on the process. Therefore, by repairing, rather than removing outlier behavior, the quality of discovered process models improves.

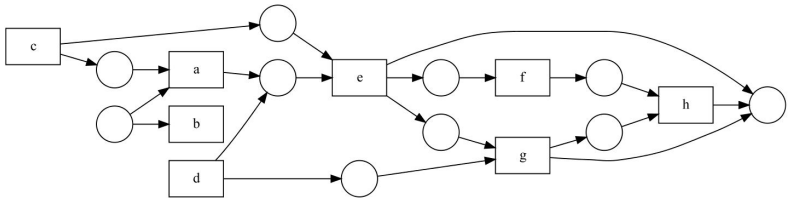
A simple example event log with some outlier behavior is shown in Tab. 1. In this event log there are 74 events that belong to 11 traces. Except for the first *variant*, i. e., unique behavior of a process instance, each variant has just one corresponding trace. This is very common in several application domains, e. g., the medical treatment process (Rebuge and Ferreira 2012). The first three traces contain no outlier behavior. However, the other seven variants (row 4 to 10) have different types of outlier behavior. For example, in the fourth and fifth rows of Tab. 1, the activities "*h*" and "*a*" are missing, respectively.

Tab. 1: Event log with 11 *traces and* 10 *different trace-variants.*

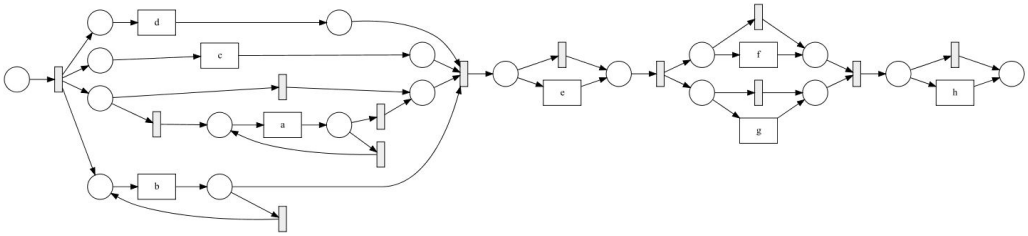
Row	*Variant*	*Frequency*
1	$\langle a, b, c, d, e, f, h\rangle$	2
2	$\langle a, b, d, c, e, f, h\rangle$	1
3	$\langle a, b, c, d, e, g, h\rangle$	1
4	$\langle a, b, d, c, e, g\rangle$	1
5	$\langle b, d, c, e, g, h\rangle$	1
6	$\langle a, b, c, d, e\rangle$	1
7	$\langle a, b, c, d, g, h\rangle$	1
8	$\langle a, b, b, c, d, e, f, h\rangle$	1
9	$\langle a, b, c, d, g, h\rangle$	1
10	$\langle a, b, d, c, a, e, g, f, h\rangle$	1

The results of applying various process discovery algorithms on the event log of Tab. 1 are shown in Fig. 1. Some process discovery algorithms like the Alpha Miner (Aalst et al. 2004) are sensitive to such outlier behavior and yield inferior process discovery results when applied directly to such event logs. Other process discovery algorithms, like the Inductive Miner (Leemans et al. 2013b) and the Split Miner (Augusto et al. 2019), have embedded filtering mechanisms to handle some types of outliers.

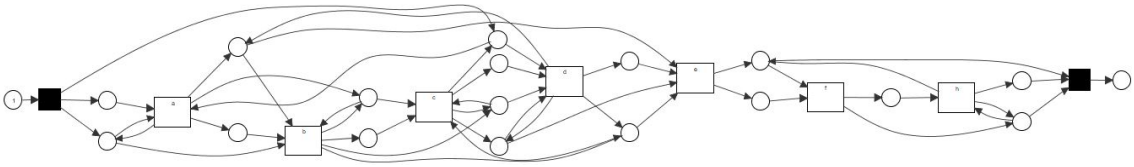
All the mentioned process discovery algorithms have difficulties to discover an accurate process model from the given event log. Also, if we apply the filtering method of Fani Sani et al. (2017) on this event log, just variants $1-5$ are retained. A resulting process model using this filtered event data combined with the Inductive Miner is shown in Fig. 1f. However, if we first repair the event log and then apply the Alpha Miner (or any other mentioned process discovery methods), we obtain a more accurate process model, i. e., as presented in Fig. 1g. This is because there is no outlier behavior in the repaired event log and the resulted



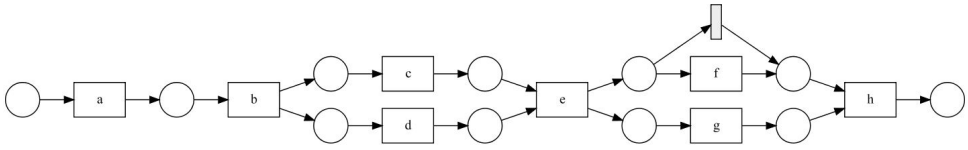
(a) Result of the Alpha Miner on the event log of Tab. 1



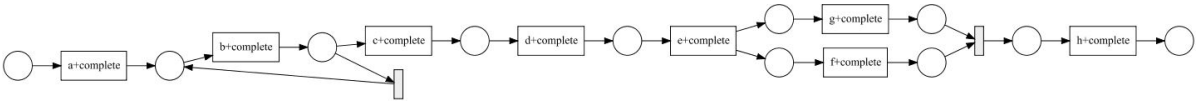
(b) Result of the Inductive Miner on the event log of Tab. 1



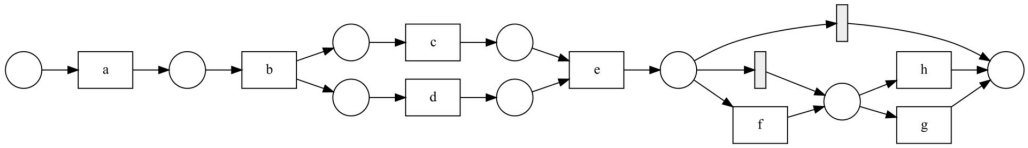
(c) Result of the ILP Miner on the event log of Tab. 1



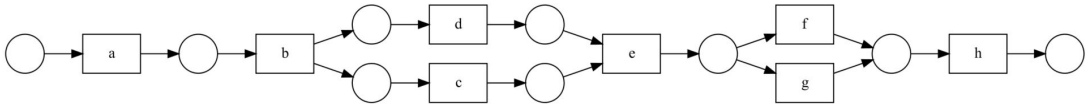
(d) Result of the Inductive Miner with integrated filtering (IMi) on the event log of Tab. 1



(e) Result of the Split Miner on the event log of Tab. 1



(f) Result of the Inductive Miner on the filtered event log by Fani Sani et al. (2017)



(g) Result of the Alpha miner on the even log of Tab. 1 after repairing outlier behavior in it.

Fig. 1: Resulting process models of applying different process discovery techniques (with and without filtering/repairing) on the event log that is presented in Tab. 1. Applying our proposed repair method outperforms discovery techniques, even when combined with state-of-the-art filtering techniques.

process model is more accurate and more understandable.

3 Related Work

Some discovery algorithms, e. g., the Alpha Miner (Aalst et al. 2004), the ILP Miner (Werf et al. 2009), and the basic Inductive Miner (Leemans et al. 2013a) were designed to incorporate all behavior in the event log. Consequently, they are sensitive to outlier behavior. Other process discovery algorithms, such as the Split Miner (Augusto et al. 2019), the Flexible Heuristic Miner (Weijters and Ribeiro 2011), the Fuzzy Miner (Günther and Aalst 2007) and extended versions of the Inductive Miner (Leemans et al. 2013b), and ILP Miner (Zelst et al. 2017) were designed to be able to handle outliers as well. However, these filtering techniques are tailored towards the internal working of the corresponding algorithms and they are not able to be used for general purpose event log preprocessing. In addition, they typically focus on a specific type of outlier, e. g., incompleteness.

Commercial tools usually use variant-based filtering methods that make it possible to only consider the most frequent trace variants, i. e., unique sequence of activities. However, the presence of parallelism and loops often hampers the applicability of such filtering techniques, because it is possible to have lots of variants that indicate the same behavior. There are also some basic filtering plug-ins developed in the PROM framework (Aalst et al. 2009) that are working based on activity frequencies and users input. But, they are usually only helpful for slicing or dicing the event log, e. g., just considering process instances that have a certain value for a specific attribute.

Outlier detection for general discrete data is addressed in some research, e. g., in Chandola et al. (2012) a survey on different methods of detecting outliers in sequential data is presented. In Gupta et al. (2014) a similar study for temporal data is presented. Also, there are some related techniques that were specifically proposed for the process mining domain. In Wang et al. (2015) and Cheng and Kumar (2015) the authors propose filtering techniques that use additional information such as training event data or business rules. In reality, providing a sufficiently complete set of training traces is impractical or even impossible.

Recently, some general purpose filtering techniques were proposed in the process mining domain. In Conforti et al. (2017) an Anomaly Free Automaton (AFA) is constructed from the whole event log. Subsequently, all non-fitting behavior, w.r.t. the AFA is removed from the event log. In Fani Sani et al. (2017), the authors propose a filtering method that detects outliers based on conditional probabilities of subsequences and their possible following activities. In Zelst et al. (2018) an adjustable on-line filtering method is proposed that detects outlier behavior for streaming events that also works based on conditional probabilities. Another filtering method that detects outlier behavior based on sequential patterns is presented in Fani Sani et al. (2018a) that performs better in event logs with the huge presence of parallel behavior. Moreover, in Mannhardt et al. (2017) the authors propose to use data attributes to detect outliers. Finally, in Tax et al. (2019) an entropy-based method is proposed to filter chaotic activities, i. e., activities that occur spontaneously at any point in the process.

All aforementioned methods, after detecting outlier behavior in a trace, aim to remove traces/behavior. As motivated in Sect. 2, repairing outlier behavior in some cases is more valuable compared to remove it. The only research in general purpose repair event logs is presented by Conforti et al. (2018) that aims to fix the time-stamps and change the order of activities that have the same time-stamp in a process instance. The order of activities are adjusted based on the most frequent behaviour that have been seen in the event log using an Automata. This method is not able to repair all type of outlier behavior like extra or missed activities.

In addition, Fahland and Aalst (2015) and Armas-Cervantes et al. (2017) aim to repair process models based on event logs. Moreover, Rogge-Solti et al. (2013) proposed a method that uses a reference process model to repair event

logs. However, as we aim to design a general purpose repair method, we assume there does not exist a process model as an input of our proposed algorithm.

4 Preliminaries

In this section, we present basic preliminaries and notations, used throughout the paper. We first introduce some basic process mining terminologies and notations, such as *subsequence* and *event log*, which we use in the proposed method.

4.1 Multisets and Sequences

Given a set X, a multiset M over X is a function $M: X \rightarrow \mathbb{N}_{\geq 0}$, that allows certain elements of X to appear multiple times. We write a multiset as $M = [x_1^{k_1}, x_2^{k_2}, ..., x_n^{k_n}]$, where for $1 \leq i \leq n$ we have $M(x_i) = k_i$ with $k_i \in \mathbb{N}_{>0}$. If $k_i = 1$, we omit its superscript, and, if for some $x \in X$ we have $M(x) = 0$, we omit it from the multiset notation. Also, $M = [\,]$ denotes the empty multiset, i. e., $\forall x \in X, M(x) = 0$. We let $\overline{M} = \{x \in X \mid M(x) > 0\}$, i. e., $\overline{M} \subseteq X$. The set of all possible multisets over a set X is written as $\mathcal{M}(X)$.

A sequence σ of length n over X is a function $\sigma: \{1, 2, ..., n\} \rightarrow X$, alternatively written as $\sigma = \langle x_1, x_2, ..., x_n \rangle$ where $x_i = \sigma(i)$ for $1 \leq i \leq n$. The empty sequence is written as ϵ. Concatenation of sequences σ and σ' is written as $\sigma \cdot \sigma'$. Let X^* denote the set of all possible sequences over a set X. We let function $hd: X^* \times \mathbb{N}_{\geq 0} \nrightarrow X^*$, represents the *head* of a sequence, i. e., given a sequence $\sigma \in X^*$ and $k \leq |\sigma|$, $hd(\sigma, k) = \langle x_1, x_2, .., x_k \rangle$, i. e., the sequence of the first k elements of σ. In case $k = 0$ we have $hd(\sigma, 0) = \epsilon$. Symmetrically, $tl: X^* \times \mathbb{N}_{\geq 0} \nrightarrow X^*$ represents the *tail* of a sequence and is defined as $tl(\sigma, k) = \langle x_{n-k+1}, x_{n-k+2}, ..., x_n \rangle$, i. e., the sequence of the last k elements of σ, with, again, $tl(\sigma, 0) = \epsilon$. Sequence σ' is a subsequence of sequence σ, which we denote as $\sigma' \in \sigma$, if and only if $\exists \sigma_1, \sigma_2 \in X^* (\sigma = \sigma_1 \cdot \sigma' \cdot \sigma_2)$.

Definition 1 (Subsequence Frequency) Let $\sigma = \langle x_1, .., x_n \rangle, \sigma' = \langle x'_1, .., x'_m \rangle \in X^*$. We define the occurrence frequency of σ' in σ by function $fr: X^* \times X^* \rightarrow \mathbb{N}_{\geq 0}$, formulated as follows.

$$fr(\sigma', \sigma) = |\{1 \leq i \leq n \mid \sigma' = \langle x_i, .., x_{i+m} \rangle\}| \tag{1}$$

i. e., $|\sigma_{\sigma'}| = fr(\sigma', \sigma)$.

Given the example event log presented in Tab. 1, we have $|\langle a, d, b, d, c, f, h \rangle_{\langle d \rangle}| = 2$ and $|\langle a, d, b, d, c, f, h \rangle_{\langle b, d \rangle}| = 1$.

4.2 Event Data

Event logs describe sequences of executed business process activities, typically in the context of some cases (or process instances), e. g., a customer or an order-id. Moreover, they act as the primary data source of any process mining analysis. Consider Tab. 2, in which we present a synthetic event log. The execution of an activity in context of a case is referred to an *event*. A sequence of events for a specific case is also referred to a *trace*. Thus, it is possible that multiple traces describe the same sequence of activities, yet, since events are unique, each trace itself contains different events. An example event log, adopted from Aalst (2016), is presented in Tab. 2.

Consider the events related to *Case-id* value 1. Nour *registers a request*, after which Alfredo *examines it thoroughly*. William *checks the ticket* and *checks resources*. Ali *makes decision*, Josef *sends the request to manager* and Fatima *accepts the request*. Finally, Daniel *emails the decision* to the client. This example trace is written as $\langle a, b, c, d, e, f, h \rangle$ (using short-hand activity names).

In the context of this paper, we only focus on the sequential ordering of the activities w.r.t. the different process instances, i. e., the *control-flow perspective*. In formal definitions, we omit additional data attributes such as resource information. We formally define event logs as a multiset of sequences of activities rather than a set of traces describing sequences of unique events.

Definition 2 (Event Log) Let $\mathcal{A}$ be a set of activities. An event log is a multiset of sequences over $\mathcal{A}$, i. e., $L \in \mathcal{M}(\mathcal{A}^*)$.

Tab. 2: Fragment of a fictional event log, where each line corresponds to an event. An event at least describes a case-id attribute, an activity attribute and a time-stamp attribute. The case-id attribute allows us to deduce to which process instance the event belongs, whereas the activity attribute describes the activity that was executed. The time-stamp attribute describes the time at which the event was executed, and allows us to order the events. Often more data attributes are available as well, e. g., resource information.

Case-id	*Activity*	*Resource*	*Time-stamp*
...	...	...	...
1	*register request (a)*	*Nour*	*2018-10-08:08.10*
1	*examine thoroughly (b)*	*Alfredo*	*2018-10-08:09.17*
2	*register request (a)*	*Nour*	*2018-10-08:10.14*
1	*check resources (c)*	*William*	*2018-10-08:10.23*
1	*check ticket (d)*	*William*	*2018-10-08:10.53*
2	*examine thoroughly (b)*	*Alfredo*	*2018-10-08:11.13*
1	*decide (e)*	*Ali*	*2018-04-08:12.14*
1	*Send to manager(e)*	*Josef*	*2018-10-08:13.09*
1	*accept request (f)*	*Fatima*	*2018-10-08:16.05*
1	*mail decision(h)*	*Daniel*	*2018-10-08:16.18*
3	*register request (a)*	*Nour*	*2018-10-08:17.14*
...	...	...	...

Observe that each $\sigma \in \overline{L}$ describes a *trace-variant* whereas $L(\sigma)$ describes how many traces of the form σ are present within the event log. For example, in the event log of Tab. 1, $L(\langle a, b, c, d, e, f, h\rangle) = 2$.

5 Repairing Outliers in Event Data

In this section, we present the proposed repair method. Consider Fig. 2, in which we present a schematic overview of the repair method.

In this figure, we represent behavior, i. e., subsequences of activities as a coloured block. According to the whole event log, for each process instance, we replace outlier behavior according to the context of behavior (i. e., proceeding/succeeding behavior). For example, it is more probable that among green and blue behavior red behavior is executed. So, if we observe purple behavior (the last process instance) surrounded by green and blue blocks, we are able to replace it with a red behavior. Note that, we assume that green and blue behavior appears frequently enough in the event log and the occurrence probability of execution of purple behavior is low.

The main components used in our approach are depicted in Fig. 3. Each of the three blocks represents subsequences in a trace. Based on the occurrence probability of a contextual sub-pattern (i. e., the red block) among its context, we try to detect outlier behavior and repair it. The green block is the *left context* and the blue block is the *right context* of the contextual sub-behavior. Each of the three blocks are allowed to be of different lengths.

The repair method consists of the following main steps:

1. *Frequent Context Identification*
   In this step, we identify *common pairs of behavior*, that frequently surround other, possibly infrequent, behavior.
2. *Infrequent Contextual Sub-Pattern Detection*
   In this step, we identify infrequent fragments of behavior, surrounded by a frequently occuring context, i. e., as identified in *Step 1*.
3. *Infrequent Contextual Sub-Pattern Replacement*
   In this step, we replace the infrequent behavioral patterns, identified in *Step 2*, by more likely/frequent behavior.

In the remainder of this section, we discuss each component in detail.

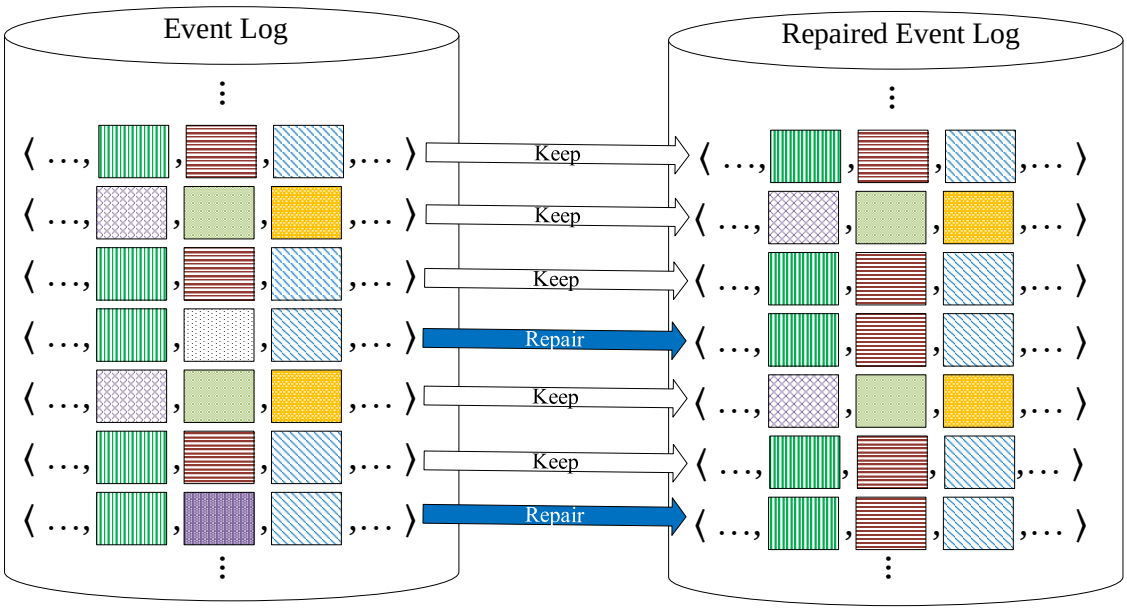


Fig. 2: Schematic overview of the proposed repair method. We identify infrequent behavioral patterns, surrounded by frequent behavior, and repair those patterns by more frequently occurring behavior.

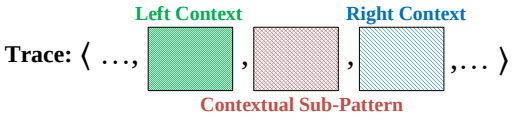


Fig. 3: Schematic overview of the different behavioral entities used throughout this paper. The behavioral context *consists of a* left- *and* right argument*, and it surrounds a* contextual sub-pattern*.*

5.1 Frequent Context Identification

In the first step of the approach, we identify whether certain pairs of behavior frequently surround other behavior. The underpinning motivation of using pairs of behavior that surround other behavior, is visualized in Fig. 4. We observe that many processes start with unique and/or similar behavior, i. e., usually an initial (group) of activities is performed in a fixed order. Subsequently, depending on the specific instance of the process, more variety is possible within the process, e. g., by means of parallel branches, loops, etc. At certain points in the process, the behavior converges again into more structured, i. e., less variable behavior, after which it diverges again.

For example, consider the process of requesting a loan at a bank. Such a process always starts with a loan request by the client. However, subsequently, more actions are possible, i. e., the bank can request more details about the client's credit history, schedule an interview with the client, or, if the client is not a customer of the bank yet, the bank first needs to create a customer profile in its information system. After these initial activities, a manager needs to verify the loan request, after which it is handed over to the risk department, i. e., the behavior converges and branches out again. The same applies for sending a semi-large packet by mail. The first step of such a process is usually a customer handing out the packet at a postal office desk. Secondly, the packet gets a unique id, and is scanned into the system. After this, depending on the size/weight of the packet, as well the destination (i. e., domestic vs. international) the activities performed for the packet start to diversity.

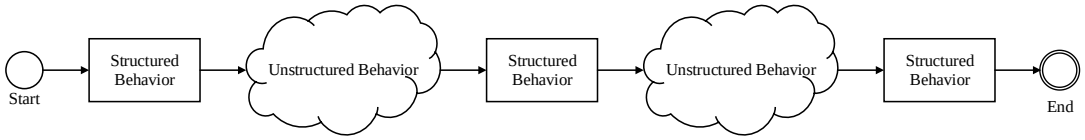


Fig. 4: Schematic overview of the typical flow of behavior in processes. Fragments of structured behavior, e. g., certain activities being always followed by the same other activity, are often followed by blocks of less structured behavior. We identify the fragments of structured behavior and aim to identify and replace parts of the unstructured behavior by the predominant behavior in that part of the process.

In the first step of the repair approach, we identify these pairs of frequent surrounding behavior, which we refer to as behavioral contexts. We define such surrounding behavior as *behavioral context*, and are interested in those contexts that occur frequently on a global event log level.

Definition 3 (Behavioral Context) Let $\mathcal{A}$ denote the universe of activities and let $L \in \mathcal{M}(\mathcal{A}^*)$ be an event log. A behavioral context c is a pair of sequences of activities, i. e., $c \in \mathcal{A}^* \times \mathcal{A}^*$. Furthermore, we define the set of behavioral contexts present in L, i. e., $\beta_L \in \mathcal{P}(\mathcal{A}^* \times \mathcal{A}^*)$, as:

$$\beta_L = \{(\sigma_l, \sigma_r) \in \mathcal{A}^* \times \mathcal{A}^* \mid \exists \sigma \in L, \sigma' \in \mathcal{A}^* (\sigma_l \cdot \sigma' \cdot \sigma_r \in \sigma)\} \tag{2}$$

For example, in trace $\langle a, b, c, d, e, f, h \rangle$, $\langle a, b \rangle$ and $\langle e \rangle$ are two subsequences that surround $\langle c, d \rangle$, hence, the pair $(\langle a, b \rangle, \langle e \rangle)$ is a behavioral context. As we are interested in those contexts that occur frequently throughout the event log, we additionally define the relative frequency of the different behavioral contexts, present in the event log.

Definition 4 (Relative Context Frequency) Let $\mathcal{A}^*$ denote the universe of activities and let $L \in \mathcal{M}(\mathcal{A}^*)$ be an event log. Let $(\sigma_l, \sigma_r) \in \mathcal{A}^* \times \mathcal{A}^*$ denote a behavioral context. We define the *relative behavioral context frequency* of context (σ_l, σ_r) in L, as a function $f_L : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}_{\geq 0}$, where:

$$f_L(\sigma_l, \sigma_r) = \frac{\sum_{\sigma \in L} \left(L(\sigma) \times \sum_{\sigma' \in \mathcal{A}^*} |\sigma_{\sigma_l \cdot \sigma' \cdot \sigma_r}| \right)}{|L|} \tag{3}$$

Observe that $f_L(\sigma_l, \sigma_r)$ represents the average number of occurrences of context (σ_l, σ_r) in the event log and is a value that is equal to or greater than 0. Note that, it is possible that $f_L(\sigma_l, \sigma_r) > 1$, due to loop structures it is possible to observe a context more than once in a trace.

The exact quantification of which behavioral contexts are relevant, is largely domain, or even event log, specific. Clearly, it is possible to use a fully automated outlier detection technique that, on the basis of the f_L-values, identifies (in)frequent behavioral contexts. Alternatively, we are able to use a user-specified threshold t_c and only include those contexts (σ_l, σ_r) with $f_L(\sigma_l, \sigma_r) \geq t_c$.

5.2 Infrequent Contextual Sub-Pattern Detection

Given a set of frequent behavioral context, we inspect the behavior that is surrounded by these contexts. In the remainder of this paper, we refer to such behavior as contextual sub-patterns. Identification of infrequent contextual sub-patterns, is relatively easy. We simply compute the empirical conditional probability of a behavioral sequence, being surrounded by a certain context.

Definition 5 (Conditional Contextual Probability) Let $\mathcal{A}^*$ denotes the universe of activities. Let $\sigma', \sigma_l, \sigma_r \in \mathcal{A}^*$ be three sequences of activities and let $L \in \mathcal{M}(\mathcal{A}^*)$ be an event log. We define the *conditional contextual probability* of σ', w.r.t., σ_l and σ_r in L, i. e., representing the empirical conditional probability of σ' being

surrounded by σ_l and σ_r in L, as a function $\gamma_L : \mathcal{A}^* \times \mathcal{A}^* \times \mathcal{A}^* \rightarrow [0, 1]$, where:

$$\gamma_L(\sigma', \sigma_l, \sigma_r) = \frac{\sum_{\sigma \in \overline{L}} \left(L(\sigma) \times |\sigma_{\sigma_l \cdot \sigma' \cdot \sigma_r}| \right)}{\sum_{\sigma \in \overline{L}} \left(L(\sigma) \times \sum_{\sigma'' \in \mathcal{A}^*} |\sigma_{\sigma_l \cdot \sigma'' \cdot \sigma_r}| \right)} \tag{4}$$

We alternatively write $P_L(\sigma' \mid \sigma_l, \sigma_r)$ to represent $\gamma_L(\sigma', \sigma_l, \sigma_r)$.

Again, we are able to use a fully automated outlier detection technique that, on the basis of the $P_L(\sigma' \mid \sigma_l, \sigma_r)$-values, identifies (in)frequent behavioral contexts. Alternatively, we are able to use a user-specified threshold.

5.3 Infrequent Contextual Sub-Pattern Replacement

In the final step of the repair procedure, we replace the *infrequent patterns* by more frequent patterns according to subsection 5.1. We identify the following replacement strategies:

- *Randomized*
  Given a frequent context and a set of all its high-probability sub-patterns, we randomly select one, according to the corresponding probabilities.
- *Maximal*
  We select the sub-pattern with the highest probability that is surrounded by the context.
- *Similarity Based*
  Considering all high-probability sub-patterns of a given context, we select the sub-pattern that has the highest similarity with the outlier sub-pattern. If there are more than one sub-patterns with the same similarity, we select the sub-pattern with the highest probability.

5.4 Repair

In this section, we combine the main steps of the approach together, in a concise algorithmic description of the proposed repair approach. To decrease the computational complexity of the proposed approach we just consider behavioral context with maximum length of subsequences equal to r and l. Therefore, we redefine the behavioral context as follows.

$$\beta_L^{r,l} = \{(\sigma_l, \sigma_r) \in \beta_L \mid 1 \leq |\sigma_l| \leq l \wedge 1 \leq |\sigma_r| \leq r\} \tag{5}$$

Observe that, the length of contexts' subsequences is not allowed to be equal to 0.

In addition, we limit the maximum length of sub-pattern to the threshold p_l. Then, relative behavioral context frequency will be as follows.

$$f_{\beta_L}^{p_l}(\sigma_l, \sigma_r) = \frac{\sum_{\sigma \in \overline{L}} \left(L(\sigma) \times \sum_{\sigma' \in \mathcal{A}^*, \sigma' \leq p_l} |\sigma_{\sigma_l \cdot \sigma' \cdot \sigma_r}| \right)}{|L|} \tag{6}$$

In the above equation, as we have σ_l and σ_r, β_L equals to $\beta_L^{r,l}$. In the same way, conditional contextual probability is adopted to the following equation:

$$\gamma_L^{p_l}(\sigma', \sigma_l, \sigma_r) = \frac{\sum_{\sigma \in \overline{L}} \left(L(\sigma) \times |\sigma_{\sigma_l \cdot \sigma' \cdot \sigma_r}| \right)}{\sum_{\sigma \in \overline{L}} \left(L(\sigma) \times \sum_{\sigma'' \in \mathcal{A}^*, |\sigma''| \leq p_l} |\sigma_{\sigma_l \cdot \sigma'' \cdot \sigma_r}| \right)} \tag{7}$$

The pseudo-code of our repair method is given in Algorithm 1. The inputs of this method are an event log (i.e, L), the maximum length of sub-pattern (i. e., $p_l \in \mathbb{Z}_{\geq 0}$), the maximum length of the left subsequence of contexts (i. e., $l \in \mathbb{N}$), the maximum length of the right subsequence of contexts (i. e., $r \in \mathbb{N}$), the minimum threshold for the relative behavioral context frequency (i. e., $T_c \in \mathbb{Z}_{\geq 0}$) and the minimum threshold for the conditional contextual probability ($0 \leq T_p \leq 1$) and it returns a repaired event log (i. e., L').

For each trace, we start from a sub-pattern length equals to 0. For the contexts, we start searching from the longest ones (equal l or r) to 1. We expect that, if a context $(\langle \sigma_{l_n}, \sigma_{l_{n-1}}, .., \sigma_{l_1} \rangle, \langle \sigma_{r_1}, .., \sigma_{r_{n-1}}, \sigma_{r_n} \rangle)$ is frequent in an event log, the context $(\langle \sigma_{l_{n-1}}, .., \sigma_{l_1} \rangle, \langle \sigma_{r_1}, .., \sigma_{r_{n-1}} \rangle)$ is also frequent. Therefore, longer contexts are more interesting and we start searching from them. According to line 17^{th} of Algorithm 1, if the corresponding context is significant (w.r.t., T_c) and the conditional contextual probability of sub-pattern σ' is not high enough (w.r.t., T_p), we replace it with a more

Algorithm 1 Repairing outlier behavior in an event log

procedure REPAIR(L, p_l, l, r, T_p ,T_c)
  $L' \leftarrow$ [] //empty multiset
  for *each* ($\sigma \in L$) **do**
    Add artificial start and end activities to σ
    for ($i \leftarrow 0$ *to* p_l) **do** //the sub-pattern
      for ($j \leftarrow l$ *to* 1) **do** // the left context
        for ($k \leftarrow r$ *to* 1) **do** //the right context
          $ind \leftarrow 0$
          if ($i + j + k + ind \leq |\sigma|$) **then** // context + sub-pattern be part of a trace σ
            $\sigma_l \leftarrow \langle \sigma_{ind}, .., \sigma_{ind+j} \rangle$
            $\sigma' \leftarrow \langle \sigma_{ind+j+1}, .., \sigma_{ind+j+i} \rangle$
            $\sigma_r \leftarrow \langle \sigma_{ind+j+i+1}, .., \sigma_{ind+j+i+k} \rangle$
            if ($f^{p_l}_{\beta_L}(\sigma_l, \sigma_r) \geq T_c \wedge \gamma^{p_l}_L(\sigma', \sigma_l, \sigma_r) \geq T_p$) **then**
              Replace(σ', (σ_l, σ_r))
              $\sigma'' \leftarrow$ *Replacement acc. strategy*
              Replace σ' *by* σ'' *in* σ
              $ind \leftarrow ind + |\sigma''|$
          $ind \leftarrow ind + 1$ // sliding through the trace
    Remove artificial start and end activities from σ
    ReparedEventLog $\leftarrow$ *Add* (*ReparedEventLog*, σ)
  return *ReparedEventLog*

suitable sub-pattern according to subsection 5.3. Please note that, after each replacement the index is increased that guarantees termination of the algorithm.

As we do not allow l and r to be equal to 0, to detect outlier behavior in the starting and the ending part of traces, for each trace in the event log, we insert an *artificial start* event and an *artificial end* event. So, based on the explained pseudo-code, these artificial events never be part of contextual sub-patterns. After repairing a trace, we omit these artificial events.

6 Evaluation

To be able to evaluate the proposed repair method, we implemented the *Repair Log* plug-in (*RL*) in the PROM framework[1] . The plug-in takes an event log as an input and returns a repaired event log. Furthermore, the user is able to specify a contextual sub-pattern probability threshold T_p, a context frequency threshold T_c, a maximum subsequence length p_l and the length of left and right sequences of both contexts.[2]

To apply our proposed method on various event logs with different thresholds and applying different process mining algorithms with various parameters, we ported the *Repair Log* (*RL*) plug-in to RAPIDPROM. RAPIDPROM is an extension of *RapidMiner* that combines scientific workflows (Bolt et al. 2016) with a range of (PROM-based) process mining algorithms. Using the implementation in RAPIDPROM, we evaluate our proposed repair method compared to three state-of-the-art process discovery algorithms and a general filtering method on different event logs.

[1] Repair Log plugin svn.win.tue.nl/repos/prom/Packages/LogFiltering

[2] To decrease the usage complexity of the plug-in, in the implementation we consider the length of both the left and the right contexts are equal.

6.1 Event logs

In our evaluation, we used both real and synthetic event logs. In this regard, 19 real event logs were used that all of them are accessible via *4tu Center for Research Data*[3] (De Leoni and Mannhardt 2015; Dongen 2012; Dongen and Borchert 2017, 2018; Mannhardt, F. (Felix) 2017; Mannhardt 2016; Steeman 2013). Basic information of these event logs is presented in Tab. 3. Some of these event logs such as $Road\_Fine$ contain high frequent variants, but in some others like $Sepci\_Cases$ there are many variants that occur only one time.

For real event logs that potentially contain outlier behavior there is no reference process model available to compare with the results of process discovery algorithms. Therefore, we designed six artificial process models with different behavior as shown in Fig. 5. These process models consequently describe different types of behavior, i. e., the first model just contains sequence constructs (*Sequence*), the second one contains sequence and many exclusive choice constructs (*Xor*), the third model contains sequence and many parallel constructs (*Parallel*), the fourth one contains sequence and loop constructs (*Loop*), the fifth process model makes it possible to skip some activities (*Skip*) and the last one contains all the previous behavioral constructs (*All*). Using these reference process models, we generated six event logs containing 5000 traces each.

We also added outlier behavior with different insertion probabilities to these original event logs. As outlier behavior we consider adding of random activities at random positions of traces, random removal of activities, and swapping of activities within traces. For example, in the $Parallel\_10$ event log, we inserted 10% of all these three aforementioned types of outlier behavior to the original event log of $Parallel$. Note that, an event log with 10% added outlier does not necessarily have 90% of clean traces from the original even log, because outlier behavior is injected on an event level. However, unlike real event logs, for each synthetic event log we have a corresponding reference process model.

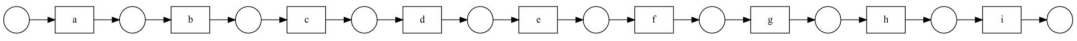
[3] https://data.4tu.nl/repository/collection:event_logs_real

6.2 Experimental Setup

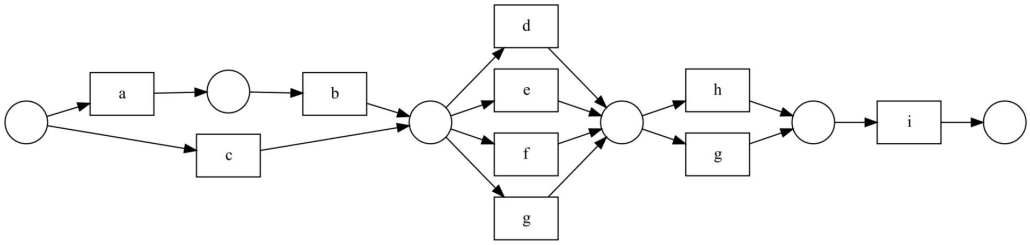
For discovering process models we apply both the Inductive Miner, the Split Miner and the ILP Miner (with and without their embedded filtering mechanism) that all guarantee to find sound process models. Soundness of discovered process models is important for our evaluation, because we are only able to compute fitness for sound process models. However, as the ILP Miner is time consuming to discover a process model, we did not apply it in all experiments. In addition, to simplify the evaluation, in all experiments we just considered the length of contexts equals to 1 to repair event logs.

We applied the process discovery algorithms on event logs with and without preprocessing. To preprocess event logs, we used both filtering and repairing methods. Therefore, as shown in Fani Sani et al. (2017), *Matrix Filter* algorithm has a good performance on event logs that contain outlier behavior and we used this method for filtering event logs. Also the proposed method was applied for repairing the event logs. As Conforti et al. (2018) is designed to repair just the order of events with the same time-stamp in a trace, we did not consider it in the evaluation. For most of the event logs that we used in the evaluation, it returns event logs similar to the input event logs. Moreover, we used preprocessed, i. e., filtered/repaired event logs just for process discovery. For conformance checking and quality assessment of discovered process models, we always used original event logs.

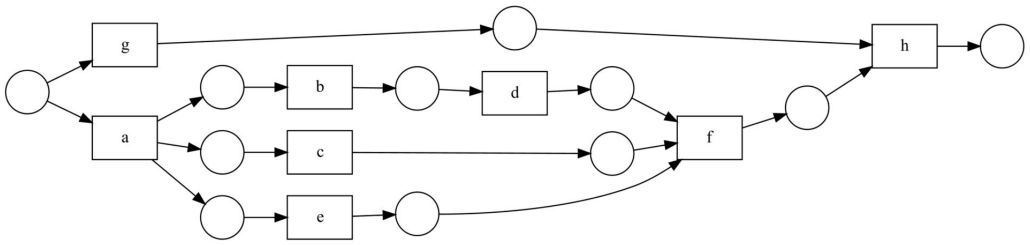
As explained, we aim to discover the best process models according to the F-Measure of these event logs using different methods. In Tab. 4, these methods and their abbreviations are explained. Two different process discovery algorithms that are the Split Miner (*S*) and the Inductive Miner (*I*) have been used with (*E*) and without (*D*) their filtering thresholds. When we used the default setting for the Inductive Miner (*ID*) we applied it without the embedded filtering mechanism. For



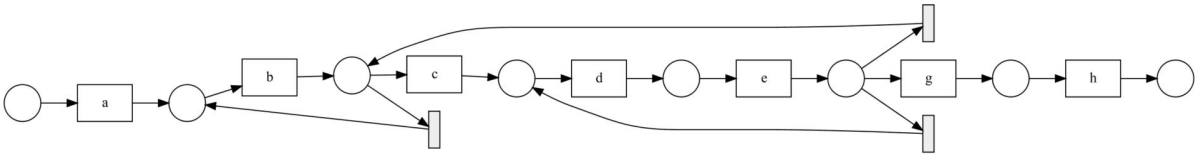
(a) A process model with sequence constructs.



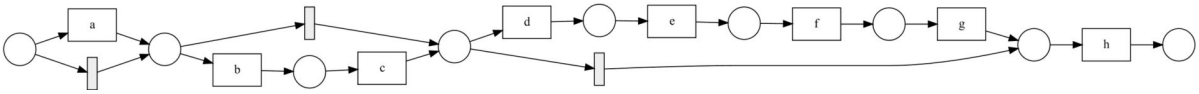
(b) A process model with exclusive choice constructs.



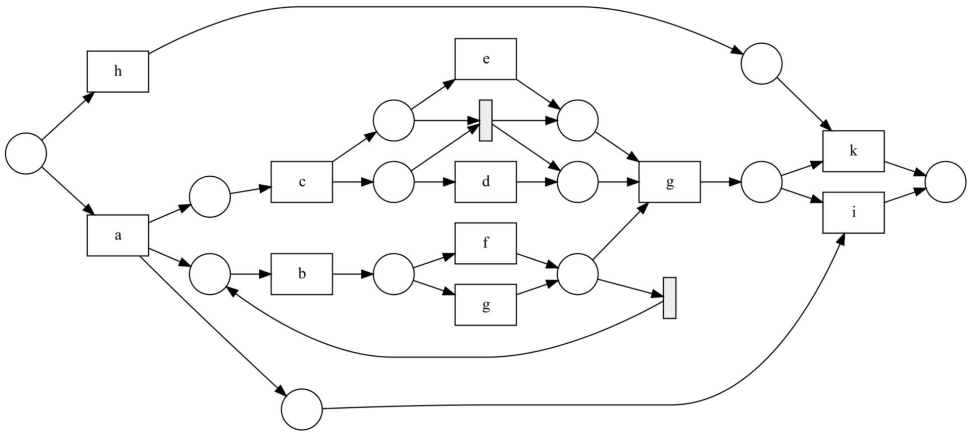
(c) A process model with parallel constructs.



(d) A process model with loop constructs.



(e) A process model with possibility to skip some activities.



(f) A process model with all above constructs.

Fig. 5: The reference process models that are used for generating synthetic event logs.

Tab. 3: Details of the real event logs that are used in the experiments.

Event Log	*Activity Count*	*Trace Count*	*Event Count*	*Variant Counts*
BPIC_2012_Application	10	13087	60849	17
BPIC_2012_Offer	7	5015	31244	168
BPIC_2012_Workflow	6	9658	72413	2263
BPIC_2013_CP	7	1487	6660	327
BPIC_2013_INC	13	7554	65533	2278
BPIC_2013_OP	5	819	2351	182
BPIC_2017_Application	10	31509	239595	102
BPIC_2017_Offer	8	42995	193849	16
BPIC_2017_Workfolow	8	31500	128227	406
BPIC_2018_Control	7	43808	161296	59
BPIC_2018_Department	6	29297	46669	349
BPIC_2018_Entitlement	40	15260	293245	2560
BPIC_2018_Financial	36	13087	262200	4366
BPIC_2018_Inspection	26	5485	197717	3190
BPIC_2018_Parcel	10	14750	132963	3615
BPIC_2018_Reference	6	43802	128554	515
Hospital_Billing	18	100000	451359	1020
Road_Fines	11	150370	561470	231
Sepsis_Cases	16	1050	15224	846

the extended version of the Inductive Miner (*IE*), we applied different filtering thresholds from 0 to 1 with steps of 0.05. For the default Split Miner (*SD*) method, we used $\epsilon = 0.1$ and $\eta = 0.4$ that are the default values of this method and for the extended version of this algorithm (SE) we used ϵ and η from 0 to 1 in 9 steps (100 different combinations). We applied these algorithms on the original event logs (*N*), filtered event logs (*F*), and repaired event logs using our proposed method (*R*). To repair and filter event logs we used threshold from 0 to 1 with steps of 0.05 and subsequence length equals to 2, 3 and 4. For instance, the abbreviation *SER* refers to applying the Split Miner with different filtering thresholds on the repaired event log.

Tab. 4: The abbreviations of twelve different methods that are used for discovering process models.

Process Discovery	Embedded Filtering	*Preprocessing*	*Abbreviation*
Inductive Miner	No	Nothing	*IDN*
	Yes	Filter	*IDF*
	No	Repair	*IDR*
	Yes	Nothing	*IEN*
	No	Filter	*IEF*
	Yes	Repair	*IER*
Split Miner	No	Nothing	*SDN*
	Yes	Filter	*SDF*
	No	Repair	*SDR*
	Yes	Nothing	*SEN*
	No	Filter	*SEF*
	Yes	Repair	*SER*

6.3 Evaluation Metrics

To evaluate discovered process models, we use *fitness*, *precision* and *complexity* metrics. Fitness computes how much behavior in the event log is also described by the process model. A fitness value equal to 1, indicates that all the behavior in the event log is described by the process model. Precision measures how much of behavior, that is described by the process model, is also present in the event log. A low precision value means that the process model allows for much more behavior compared to the event log. Note that, there is a trade-off between these measures (Weerdt et al. 2011). Sometimes, putting aside a small amount of behavior causes a slight decrease in the fitness value, whereas the precision value increases much more. Therefore, we use the F-Measure that combines fitness and precision as follows.

$$\text{F-Measure} = \frac{2 \times \text{Precision} \times \text{Fitness}}{\text{Precision} + \text{Fitness}} \qquad (8)$$

Moreover, there are some complexity metrics that measure the understandability of a process model (Mendling 2008). The complexity metrics relate the understandability of a process model to some parameters like the size (number of the nodes

and arcs), control flow complexity (branchings and gateways) and the structuredness of it.

6.4 Evaluation Results

To evaluate our proposed method we conducted several experiments with both real and artificial event logs. We first explain the experimental results on real event logs, after which the results of using synthetic event logs are discussed.

6.4.1 Experiments with Real Data

In the first experiment, we aim to investigate the usefulness of applying the proposed method on improving the F-Measure of discovered process models. Tab. 5 shows the best obtained F-Measure values of applying different methods (i. e., described in Tab. 4) on real event logs. The results show that preprocessing the event log using both the filtering and repairing methods improves the F-Measure of discovered process models. They usually improve the F-Measure by sacrificing a bit in fitness and increasing the precision. This improvement is usually higher if the $\frac{Trace\#}{Variant\#}$ is lower. So, if the number of unique traces is higher we expect the improvement of preprocessing to be higher. Therefore, the improvements in F-Measure of discovered process models for $Hospital\_Billing$ and $Sepsis$ event logs are higher, specifically when the Split Miner was used. The results indicate that usually the combination of the preprocessing methods and filtering mechanisms that are embedded in the process discovery algorithm leads to a higher or equal F-Measure values. In addition, by comparison of the preprocessing methods results (i. e., *IEF* vs. *IER* and *SEF* vs. *SER*), we found that for most event logs, applying the repair method increases the value of F-Measure more.

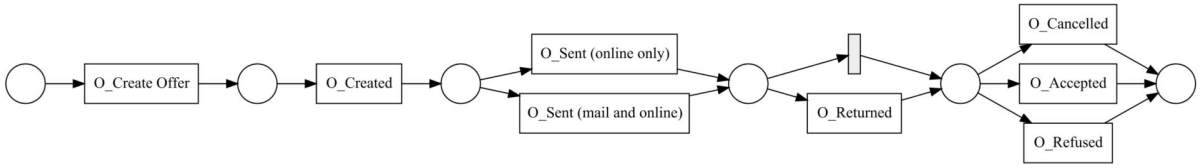
As shown in Tab. 6, for most event logs, using the embedded filtering mechanism in process discovery algorithms on the *repaired event logs* (i. e., *SER* and *IER*) results in the process model with the highest F-Measure. For 14 event logs out of 19, the best process models for the Inductive Miner is obtained when the repaired event logs are used.[4] . Thus, we conclude that preprocessing event logs, specially using our method improves the results of process discovery algorithms.

Note that, for some event logs such as the $Sepsis\_Cases$, neither repairing nor filtering helps us to have an outstanding process model, i.e, obtaining the high F-Measure value. It relates to the fact that the discovered process model of this event log is not precise and there is some behavior that is possible to happen anywhere during the execution of the process. In general, having any *parallel* and/or *skip* behavior in the model causes a decrease at the precision value of the process model, even when it depicts the behavior in the event log correctly. We have shown the best discovered process model for this event log to a business expert and he expressed that the resulted process model is acceptable (but, not perfect).

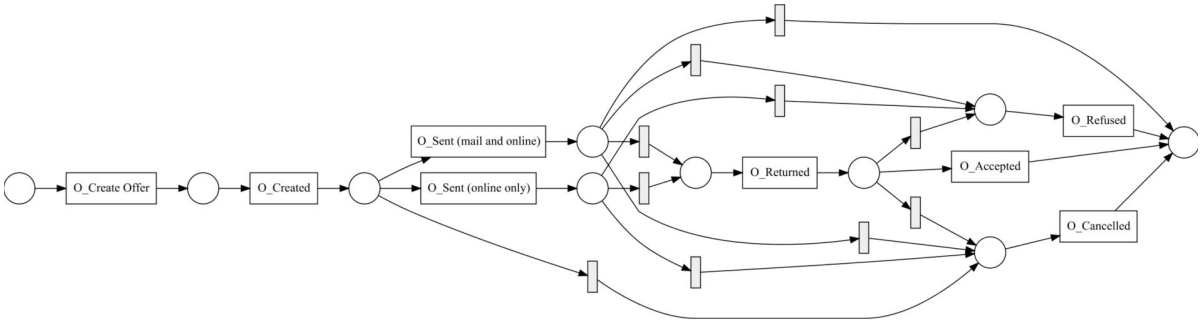
Fig. 6 shows the best process models of different methods resulted from the above experiment for the $BPIC\_2017\_Offer$ event log. Fig. 6f and Fig. 6e which are discovered on repaired event logs have less complexity and high F-Measure. With respect to their F-Measure we are able to say that they are the best process models for this event log among the other methods. These process models are depicting the same behavior, however, in Fig. 6f, there is an unnecessary silent transition at the end of the process model. In addition, from these results we found that preprocessing the event log not only improves the F-Measure of the process models, but the understandability of the process models is increased as well.

In the second experiment, to evaluate the helpfulness of our proposed method in reducing the complexity, we discovered process models on repaired event logs using the ILP Miner (Zelst et al. 2015). Please note that, as the ILP miner is very time consuming, we repaired real event logs with just six different thresholds and consider the process models with the highest F-Measure. As shown in Fig. 7, the F-Measures of discovered

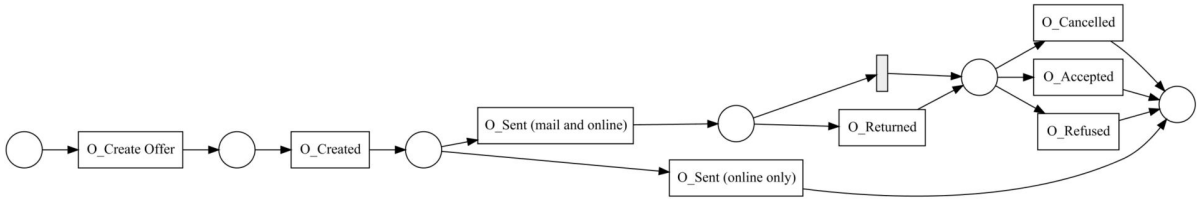
[4] For this comparison we used more accurate decimals values compared to Tab. 5



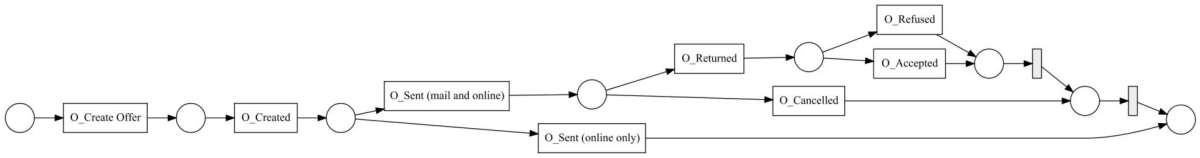
(a) Results of Applying the Inductive Miner on the original event log (IEN).



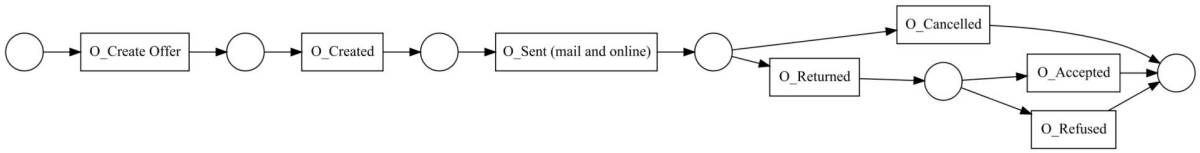
(b) Results of Applying the Split Miner on the original event log (SEN).



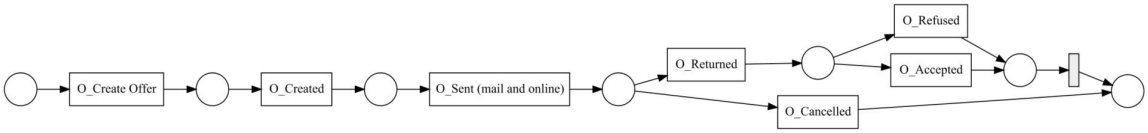
(c) Results of Applying the Inductive Miner on the filtered event log (IEF).



(d) Results of Applying the Split Miner on the filtered event log (SEF).



(e) Results of Applying the Inductive Miner on the repaired event log (IER).



(f) Results of Applying the Split Miner on the repaired event log (SER).

Fig. 6: The best discovered process models on BPIC_2017_Offer with and without preprocessing. For preprocessing we used filtering and repairing methods.

Tab. 5: Best F-Measure of applying the twelve different methods on some real event logs.

Event Log	IDN	IDF	IDR	IEN	IEF	IER	SDN	SDF	SDR	SEN	SEF	SER
BPIC_2012_Application	0.842	0.917	0.971	0.93	0.94	0.976	0.961	0.97	0.966	0.977	0.975	0.977
BPIC_2012_Offer	0.533	0.896	0.907	0.884	0.925	0.907	0.976	0.978	0.976	0.976	0.978	0.978
BPIC_2012_Work	0.447	0.667	0.589	0.76	0.775	0.817	0.739	0.764	0.806	0.739	0.769	0.809
BPIC_2013_CP	0.671	0.929	0.929	0.784	0.929	0.929	0.686	0.929	0.929	0.713	0.929	0.929
BPIC_2013_Inc	0.827	0.925	0.946	0.695	0.943	0.958	0.68	0.789	0.908	0.774	0.79	0.908
BPIC_2013_OP	0.735	0.877	0.882	0.858	0.877	0.923	0.797	0.877	0.866	0.816	0.877	0.866
BPIC_2017_APPlication	0.904	0.906	0.969	0.965	0.965	0.976	0.866	0.927	0.948	0.921	0.959	0.976
BPIC_2017_Offer	0.878	0.93	0.987	0.983	0.984	0.987	0.953	0.976	0.987	0.97	0.987	0.987
BPIC_2017_Workflow	0.716	0.905	0.852	0.905	0.909	0.906	0.918	0.918	0.92	0.921	0.921	0.921
BPIC_2018_Control	0.925	0.991	0.999	0.955	0.981	0.999	0.996	1	0.999	0.996	1	0.999
BPIC_2018_Department	0.796	0.892	0.894	0.85	0.882	0.894	0.882	0.89	0.889	0.889	0.893	0.891
BPIC_2018_Entitlement	0.528	0.929	0.751	0.649	0.943	0.924	0.68	0.929	0.929	0.867	0.929	0.939
BPIC_2018_Financial	0.537	0.817	0.707	0.729	0.862	0.791	~	0.856	0.691	0.826	0.868	0.86
BPIC_2018_Inspection	0.657	0.727	0.748	0.726	0.734	0.86	0.783	0.784	0.878	0.83	0.806	0.881
BPIC_2018_Parcel	0.579	0.915	0.945	0.737	0.927	0.956	0.872	0.916	0.955	0.913	0.916	0.955
BPIC_2018_Reference	0.855	0.991	0.997	0.931	0.991	0.997	0.992	0.997	0.997	0.992	0.997	0.997
Hospital_Billing	0.817	0.912	0.871	0.876	0.921	0.961	0.934	0.988	0.98	0.984	0.988	0.991
Road_Traffic	0.595	0.957	0.958	0.776	0.951	0.952	0.693	0.933	0.956	0.924	0.933	0.976
Sepsis	0.621	0.763	0.817	0.651	0.787	0.834	0.379	0.727	0.777	0.671	0.73	0.783

Tab. 6: The number of event logs in which the preprocessing method results in the best process model according to F-Measure.

Method	DN	DF	DR	EN	EF	ER
Inductive Miner	0	1	5	0	5	14
Split Miner	0	5	4	2	9	13

process models are highly increased when we repaired the event logs beforehand. The difference between F-Measure values in this figure is high because in ILP miner no filtering method is used to handle outliers. The ILP miner always guarantees the Fitness value equal 1 for the original event log. However, it usually results in very complex process models that are not understandable. Because this algorithm aims to depict the outlier behavior in the process model (to have the fitness value equals to *1*). In Tab. 7, we compare the complexity of discovered process models using three different complexity metrics. The -1 value in the *Structuredness* column means that the corresponding process model is not sound. In the *Cyclomatic* column, the -1 value means that the process model is too complex and we are not able to compute an accurate value according to this metric. Considering these results, we found that our proposed method is able to decrease the complexity of the discovered process models and, therefore, increase their understandability.

In addition, the Cardoso values of process models when we used filtering and repairing event logs for the Inductive Miner and the Split Miner are given in Tab. 8. In almost all of the event logs, both the preprocessing methods by removing infrequent behavior reduce the complexity of discovered process models.

6.4.2 Experiments with Synthetic Data

In the previous experiments, to compute F-Measures the original event log that possibly contains outliers is used as a ground truth. Here

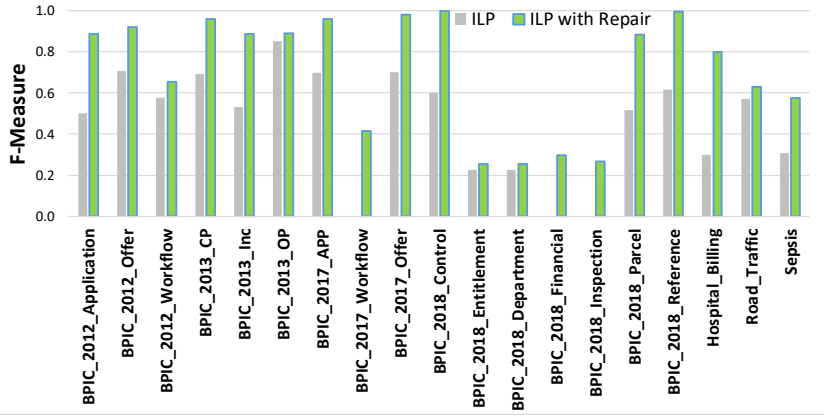


Fig. 7: The best F-Measure of process models that are discovered by the ILP miner.

Tab. 7: The complexity of discovered process models using the ILP miner with/without repairing the real event logs.

	ILP			*ILP with Repair*		
Event Logs	*Cardoso*	*Cyclomatic*	*Structuredness*	*Cardoso*	*Cyclomatic*	*Structuredness*
BPIC_2012_App	35	-1	-1	14	12	1101
BPIC_2012_Offer	20	8	1080	8	8	9.5
BPIC_2012_Workflow	16	8	1400	5	7	210
BPIC_2013_CP	15	9	1395	4	4	5
BPIC_2013_Inc	11	15	3000	4	4	5
BPIC_2013_OP	15	7	665	10	5	250
BPIC_2017_APP	27	15	3900	13	10	720
BPIC_2017_Offer	17	12	3200	8	9	36
BPIC_2017_Workflow	259	-1	-1	29	9	2475
BPIC_2018_Control	7	-1	-1	7	6	60
BPIC_2018_Department	6	6	75	3	8	16
BPIC_2018_Entitlement	71	22	23980	66	19	15865
BPIC_2018_Financial	438	-1	-1	150	-1	-1
BPIC_2018_Inspection	56	-1	-1	22	-1	5950
BPIC_2018_Parcel	21	-1	-1	7	8	90
BPIC_2018_Reference	6	-1	-1	6	5	50
Hospital_Billing	9	22	5400	5	20	1900
Road_Traffic	15	-1	-1	10	-1	-1
Sepsis	88	-1	14760	13	-1	-1

Tab. 8: The Cardoso values of discovered process models using different preprocessing methods with default settings.

Process Discovery	*Inductive Miner*			*Split Miner*		
Event Log/ preprocess	*Nothing*	*Filter*	*Repair*	*Nothing*	*Filter*	*Repair*
BPIC_2012_App	14	14	14	25	19	17
BPIC_2012_Offer	17	12	8	9	10	10
BPIC_2012_Workflow	22	18	21	33	29	33
BPIC_2013_CP	17	4	4	11	4	4
BPIC_2013_Inc	10	9	4	15	13	7
BPIC_2013_OP	19	4	4	13	13	2
BPIC_2017_APP	21	12	10	30	20	14
BPIC_2017_Offer	5	6	6	18	9	7
BPIC_2017_Workflow	30	16	33	43	43	43
BPIC_2018_Control	17	8	4	19	8	12
BPIC_2018_Department	26	16	18	22	16	23
BPIC_2018_Entitlement	23	12	19	189	176	80
BPIC_2018_Financial	25	16	43	157	145	150
BPIC_2018_Inspection	23	11	25	59	58	31
BPIC_2018_Parcel	31	12	8	31	18	16
BPIC_2018_Reference	20	8	3	20	8	15
Hospital_Billing	41	14	25	83	43	69
Road_Traffic	17	7	16	86	62	76
Sepsis	20	12	16	173	13	90

similar to the previous experiment, we applied the 12 different methods (that are explained in Tab. 4) on synthetic event logs to discover the best process models w.r.t., F-Measure. For synthetic event logs, we have event logs without outliers and we used them for computing the F-Measure. Results of this experiment are given in Tab. 9.

As shown in this table, for all event logs if we insert some noisy behavior (even just 5%), the Inductive Miner and the Split Miner have problems to discover a process model with a high F-Measure value. It means that the embedded noise filtering mechanisms in these algorithms are not able to deal with all types of outlier behavior. As shown in Tab. 9 by the ~ symbol, because of the high complexity of discovered process models, we are not able to compute the F-Measure for the discovered models of some event logs. It is interesting that for most of the event logs like *All* and *Parallel* without noisy behavior (i. e., 0%) the F-Measure of the discovered process models do not equal to 1. It is because of the precision algorithm that does not return value 1 even for the reference process model when we have *parallel* and *Xor* behavior. Furthermore, both filtering and repairing of event logs increase the F-Measure of the discovered models for both process discovery algorithms. For the Split Miner, in most of the cases, using the proposed repair method (*SER*) results the best process models. However, for the Inductive Miner; especially, when we have *loop* and *parallel* behavior, using the filtering method increases the F-Measure more. Because the Split Miner has an internal mechanism to handle some parallel behavior in event logs that the Inductive Miner does not have it. Therefore, when we have parallel and loop behavior in an event log and we want to use the Inductive miner, it is proposed to use the filtering methods instead of the repairing method.

We observe again that the combination of filtering mechanisms in the process discovery algorithms and the proposed method results in the best process model. The reason for this is related to the fact that our method is able to detect and repair outlier behavior locally. Even for the event logs that contain no noisy behavior (specially when there is heavy presence of *Xor* behavior), repairing an event log improves the process model according to the F-Measure metric.

Finally, note that filtering methods achieve the best results by removing a lot of behavior from event logs. The percentages of remaining traces in each event log for the best process model (according to F-Measure) in *IEF* and *SEF* are given in Tab. 10. In some event logs (e. g., $Skip\_05$), a few percentage of outlier behavior causes that the best process model is discovered just with around 10% of the traces. Because of using random noise generation and different parameters values of filtering and process discovery algorithms, and just showing the results of the situation with the highest F-Measure, we could not analyze all patterns in this table. However, results show that using filtering, we remove lots of traces to achieve the best process models. Even, for some clean event logs, the best process model is discovered by removing more than 35% of traces of the original event log. However, in the repair method all the traces remain in the event log, but they may be modified. As explained in Sect. 2, in some domains repairing the process instances are more valuable compared to just filtering them out. In addition, it is possible that by repairing event logs, we remove/hide much of normal infrequent behavior.

As explained in subsection 6.2, here, for all methods we used a grid search on different parameters and show the best obtained result. However, in reality like other state-of-the-art process mining specific data cleansing methods, adjusting these thresholds is a challenging task for users.

7 Conclusion

Process mining provides insights into the actual execution of business processes, by exploiting available event data. Unfortunately, many process mining algorithms are designed to work under the assumption that the input data is free of outliers. However, real event logs contain outlier behavior (noise/infrequent) which typically leads

Tab. 9: Best F-Measure of applying the Split Miner and the Inductive Miner on synthetic event logs that contain different percentages of outlier behavior.

Outlier%	*IDN*	*IDF*	*IDR*	*IEN*	*IEF*	*IER*	*SDN*	*SDF*	*SDR*	*SEN*	*SEF*	*SER*
Sequence												
0%	1	1	1	1	1	1	1	1	1	1	1	1
5%	0.418	1	1	0.53	1	1	0.652	1	0.994	0.828	1	0.994
10%	0.213	0.637	0.947	0.379	1	1	0.629	0.947	0.988	0.81	0.947	0.988
20%	0.19	0.637	0.9	0.379	1	1	0.596	0.947	0.975	0.81	0.947	0.975
50%	0.209	0.653	0.775	0.379	1	1	0.611	0.947	0.951	0.81	0.947	0.956
Xor/Choice												
0%	0.506	0.68	0.939	0.506	0.905	0.939	0.94	0.94	0.95	0.956	0.956	0.97
5%	0.495	0.806	0.815	0.619	0.909	0.908	0.7	0.943	0.97	0.908	0.953	0.973
10%	0.488	0.806	0.808	0.548	0.909	0.908	0.672	0.96	0.965	0.899	0.96	0.968
20%	0.34	0.82	0.872	0.333	0.867	0.908	0.606	0.962	0.945	0.893	0.962	0.945
50%	0.312	0.803	0.847	0.309	0.844	0.945	0.608	0.916	0.911	0.877	0.922	0.916
Parallel												
0%	0.977	0.977	0.977	0.977	0.977	0.977	0.949	0.949	0.956	0.96	0.969	0.965
5%	0.314	0.697	0.781	0.447	0.902	0.88	0.685	0.889	0.941	0.853	0.906	0.95
10%	0.291	0.615	0.71	0.456	0.897	0.855	0.658	0.741	0.942	0.853	0.907	0.942
20%	0.282	0.537	0.747	0.456	0.854	0.807	0.654	0.856	0.921	0.856	0.879	0.921
50%	0.285	0.592	0.728	0.419	0.854	0.863	0.641	0.856	0.897	0.839	0.869	0.901
Loop												
0%	0.892	0.901	0.892	0.919	0.95	0.919	0.94	0.94	0.942	0.94	0.959	0.959
5%	0.39	0.901	0.871	0.451	0.95	0.919	0.666	0.94	0.956	0.858	0.959	0.956
10%	~	0.826	0.871	0.375	0.95	0.919	0.641	0.94	0.951	0.858	0.959	0.952
20%	~	0.883	0.857	0.375	0.902	0.902	0.615	0.94	0.939	0.853	0.959	0.939
50%	~	0.883	0.851	0.375	0.919	0.902	0.636	0.944	0.904	0.853	0.944	0.918
Skip												
0%	1	1	1	1	1	1	0.881	0.911	0.881	0.906	0.911	0.943
5%	0.295	0.821	0.765	0.346	0.899	0.95	0.697	0.904	0.99	0.858	0.948	0.992
10%	0.296	0.805	0.621	0.346	0.877	0.95	0.687	0.904	0.984	0.858	0.926	0.985
20%	0.295	0.675	0.695	0.346	0.898	0.931	0.699	0.818	0.955	0.858	0.901	0.962
50%	0.307	0.663	0.65	0.353	0.91	0.894	0.689	0.795	0.924	0.863	0.883	0.937
All												
0%	~	0.731	0.822	0.882	0.889	0.867	0.866	0.863	0.886	0.883	0.879	0.889
5%	~	0.845	0.815	0.566	0.875	0.867	0.65	0.862	0.883	0.828	0.874	0.891
10%	~	0.815	0.791	0.563	0.87	0.867	0.651	0.862	0.883	0.828	0.873	0.893
20%	~	0.815	0.791	0.563	0.87	0.867	0.647	0.862	0.879	0.828	0.872	0.885
50%	~	0.762	0.828	0.563	0.854	0.867	0.648	0.869	0.863	0.828	0.869	0.872

Tab. 10: The percentage of remaining traces in event logs when filtering method is used as a preprocessing step.

Method	*IEF*					*SEF*				
Outlier %	*00*	*05*	*10*	*20*	*50*	*00*	*05*	*10*	*20*	*50*
Sequence	100%	95%	89%	75%	53%	100%	95%	89%	74%	52%
Xor	51%	43%	41%	77%	30%	100%	100%	89%	77%	59%
Parallel	100%	94%	90%	53%	41%	98%	96%	92%	86%	78%
Loop	60%	58%	37%	75%	52%	100%	96%	90%	75%	52%
Skip	100%	37%	29%	30%	28%	38%	7%	96%	83%	28%
All	65%	62%	12%	10%	13%	100%	41%	41%	4%	39%

to inaccurate/unusable process mining results. Detecting such behavior in event logs and correcting it, helps to improve process mining results, e. g., discovered process models.

To address this problem, we propose a method that repairs event logs. It uses the occurrence frequency of a control-flow-oriented context pattern and the probabilities of different subsequences appearing in the middle of it to detect outlier behavior. If such probability is lower than a given threshold, the subsequence is substituted with a more probable one according to the context.

To evaluate the proposed method, we have developed a plug-in in the PROM platform and also ported to RAPIDPROM. As presented, we have applied this method on several real event logs, and compared it with other state-of-the-art process mining specific data cleansing methods. Additionally, we applied our method on synthetic event logs. The results indicate that the proposed repair approach is able to detect and modify outlier behavior and consequently is able to help process discovery algorithms to return models that better balance between different behavioral quality measures. Furthermore, using these experiments we show that our repair method outperforms process discovery algorithms like the Inductive Miner, the Split Miner and ILP Miner as the best state-of-art process discovery algorithms. The results show that the proposed method is able to reduce the complexity of process models and, consequently, improves the understandability of them.

As future work, we want to combine different preprocessing methods. We also plan to develop techniques to automatically set adjustable filtering and repairing parameters based on characteristics of the input event log to guide users and speed-up analysis. Moreover, we just used the *sequence* abstraction in this research. As future work, it is possible to use any other abstractions such as *multi-set* and *set* for both of the contextual sub-patterns and contexts.

References

van der Aalst W. M. P. (2011) Using Process Mining to Bridge the Gap between BI and BPM. In: IEEE Computer 44(12), pp. 77–80

van der Aalst W. M. P. (2016) Process Mining - Data Science in Action, Second Edition. Springer

van der Aalst W. M. P., Bolt A., van Zelst S. J. (2017) RapidProM: Mine Your Processes and Not Just Your Data. In: CoRR abs/1703.03740 http://arxiv.org/abs/1703.03740

van der Aalst W. M. P., van Dongen B. F., Günther C. W., Rozinat A., Verbeek E., Weijters T. (2009) ProM: The Process Mining Toolkit. In: http://ceur-ws.org/Vol-489/paper3.pdf

van der Aalst W. M. P., Weijters T., Maruster L. (2004) Workflow Mining: Discovering Process Models from Event Logs. In: IEEE Trans. Knowl. Data Eng. 16(9), pp. 1128–1142

van der Aalst W., Adriansyah A., De Medeiros A. K. A., Arcieri F., Baier T., Blickle T., Bose J. C., Van Den Brand P., Brandtjen R., Buijs J., et al. (2011) Process mining manifesto. In: International Conference on Business Process Management. Springer, pp. 169–194

Andrews R., Suriadi S., Ouyang C., Poppe E. (2018) Towards Event Log Querying for Data Quality - Let's Start with Detecting Log Imperfections. In: On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part I, pp. 116–134

Armas-Cervantes A., van Beest N. R. T. P., Rosa M. L., Dumas M., Raboczi S. (2017) Incremental and Interactive Business Process Model Repair in Apromore. In: Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling (BPM 2017), Barcelona, Spain, September 13, 2017.

Augusto A., Conforti R., Dumas M., Rosa M. L., Polyvyanyy A. (2019) Split miner: automated discovery of accurate and simple business process models from event logs. In: Knowl. Inf. Syst. 59(2), pp. 251–284

Bolt A., de Leoni M., van der Aalst W. M. P. (2016) Scientific workflows for process mining: building blocks, scenarios, and implementation. In: STTT 18(6), pp. 607–628

Chandola V., Banerjee A., Kumar V. (2012) Anomaly Detection for Discrete Sequences: A Survey. In: IEEE Trans. Knowl. Data Eng. 24(5), pp. 823–839

Cheng H.-J., Kumar A. (2015) Process Mining on Noisy Logs —Can Log Sanitization Help to Improve Performance? In: Decision Support Systems 79, pp. 138–149

Conforti R., La Rosa M., ter Hofstede A. (2018) Timestamp Repair for Business Process Event Logs.

Conforti R., Rosa M. L., ter Hofstede A. H. M. (2017) Filtering Out Infrequent Behavior from Business Process Event Logs. In: IEEE Trans. Knowl. Data Eng. 29(2), pp. 300–314

De Leoni M., Mannhardt F. (2015) Road traffic fine management process. In: Eindhoven University of Technology. Dataset

van Dongen B. (2012) BPI Challenge 2012, Event log of a loan application process

van Dongen B., Borchert F. (2017) BPI Challenge 2017. Eindhoven University of Technology. Dataset.

van Dongen B., Borchert F. (2018) BPI Challenge 2018. Eindhoven University of Technology. Dataset.

Fahland D., van der Aalst W. M. P. (2015) Model repair - aligning process models to reality. In: Inf. Syst. 47, pp. 220–243

Fani Sani M., van Zelst S. J., van der Aalst W. M. P. (2017) Improving Process Discovery Results by Filtering Outliers Using Conditional Behavioural Probabilities. In: Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers, pp. 216–229

Fani Sani M., van Zelst S. J., van der Aalst W. M. P. (2018a) Applying Sequence Mining for Outlier Detection in Process Mining. In: On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part II, pp. 98–116

Fani Sani M., van Zelst S. J., van der Aalst W. M. P. (2018b) Repairing Outlier Behaviour in Event Logs. In: Business Information Systems - 21st International Conference, BIS 2018, Berlin, Germany, July 18-20, 2018, Proceedings, pp. 115–131

Günther C. W., van der Aalst W. M. P. (2007) Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In: Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings, pp. 328–343

Gupta M., Gao J., Aggarwal C. C., Han J. (2014) Outlier Detection for Temporal Data: A Survey. In: IEEE Trans. Knowl. Data Eng. 26(9), pp. 2250–2267

Hernández M. A., Stolfo S. J. (1998) Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. In: Data Min. Knowl. Discov. 2(1), pp. 9–37

Leemans S. J. J., Fahland D., van der Aalst W. M. P. (2013a) Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings, pp. 311–329

Leemans S. J. J., Fahland D., van der Aalst W. M. P. (2013b) Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In: Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, pp. 66–78

Mannhardt, F. (Felix) (2017) Hospital Billing - Event Log. https://data.4tu.nl/repository/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741

Mannhardt F. (2016) Sepsis cases-event log. Eindhoven University of Technology

Mannhardt F., de Leoni M., Reijers H. A., van der Aalst W. M. P. (2017) Data-Driven Process Discovery - Revealing Conditional Infrequent Behavior from Event Logs. In: Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings, pp. 545–560

Mendling J. (2008) Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness. Lecture Notes in Business Information Processing Vol. 6. Springer https://doi.org/10.1007/978-3-540-89224-3

Rebuge Á., Ferreira D. R. (2012) Business process analysis in healthcare environments: A methodology based on process mining. In: Inf. Syst. 37(2), pp. 99–116

Ribeiro C. E., Zárate L. E. (2016) Data Preparation for Longitudinal Data Mining: a case study on human ageing. In: JIDM 7(2), pp. 116–129

Rogge-Solti A., Mans R., van der Aalst W. M. P., Weske M. (2013) Improving Documentation by Repairing Event Logs. In: The Practice of Enterprise Modeling - 6th IFIP WG 8.1 Working Conference, PoEM 2013, Riga, Latvia, November 6-7, 2013, Proceedings, pp. 129–144

Steeman W. (2013) BPI Challenge 2013, incidents

Tax N., Sidorova N., van der Aalst W. M. P. (2019) Discovering more precise process models from event logs by filtering out chaotic activities. In: J. Intell. Inf. Syst. 52(1), pp. 107–139

Wang J., Song S., Lin X., Zhu X., Pei J. (2015) Cleaning structured event logs: A graph repair approach. In: 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015, pp. 30–41

Weerdt J. D., Backer M. D., Vanthienen J., Baesens B. (2011) A robust F-measure for evaluating discovered process models. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France, pp. 148–155

Weijters A. J. M. M., Ribeiro J. T. S. (2011) Flexible Heuristics Miner (FHM). In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France, pp. 310–317

van der Werf J. M. E. M., van Dongen B. F., Hurkens C. A. J., Serebrenik A. (2009) Process Discovery using Integer Linear Programming. In: Fundam. Inform. 94(3-4), pp. 387–412

van Zelst S. J., van Dongen B. F., van der Aalst W. M. P. (2015) Avoiding Over-Fitting in ILP-Based Process Discovery. In: Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings, pp. 163–171

van Zelst S. J., van Dongen B. F., van der Aalst W. M. P., Verbeek H. M. W. (2017) Discovering Relaxed Sound Workflow Nets using Integer Linear Programming. In: CoRR abs/1703.06733 http://arxiv.org/abs/1703.06733

van Zelst S. J., Fani Sani M., Ostovar A., Conforti R., Rosa M. L. (2018) Filtering Spurious Events from Event Streams of Business Processes. In: Advanced Information Systems Engineering - 30th International Conference, CAiSE 2018, Tallinn, Estonia, June 11-15, 2018, Proceedings, pp. 35–52