Repairing Outlier Behaviour in Event Logs

Mohammadreza Fani Sani[1], Sebastiaan J. van Zelst[2], Wil M.P. van der Aalst[1]

[1] Process and Data Science Chair, RWTH Aachen University
Ahornstraße 55, Erweiterungsbau E2, 52056 Aachen, Germany
[2] Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{fanisani,wvdaalst}@pads.rwth-aachen.de,{s.j.v.zelst}@tue.nl

Summary. One of the main challenges in applying process mining on real event data, is the presence of noise and rare behaviour. Applying process mining algorithms directly on raw event data typically results in complex, incomprehensible, and, in some cases, even inaccurate analyses. As a result, correct and/or important behaviour may be concealed. In this paper, we propose an event data repair method, that tries to detect and repair outlier behaviour within the given event data. We propose a probabilistic method that is based on the occurrence frequency of activities in specific contexts. Our approach allows for removal of infrequent behaviour, 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 framework. Using these implementations, we conduct a collection of experiments that show that we are able to detect and modify most types of outlier behaviour in the event data. Our evaluation clearly demonstrates that we are able to help to improve process mining discovery results by repairing event logs upfront.

Key words: Process Mining · Data Cleansing · Log Repair · Event Log Preprocessing · Conditional Probability · Outlier Detection

1 Introduction

Process Mining bridges the gap between traditional data analysis techniques like data mining and business process management analysis [1]. It aims to discover, monitor, and enhance processes by extracting knowledge from event data, also referred to as *event logs*, readily available in most modern information systems [2]. In general, we identify three main branches of process mining being *process discovery*, *conformance checking*, and *process enhancement*. In process discovery, we try to discover a process model that accurately describes the underlying process captured within the event data. In conformance checking we try to assess to what degree a given process model (possibly the result of a process discovery algorithm) and event data conform to one another. Finally, process enhancement aims at improving the view on a process by improving its corresponding model based on the related event data.

Most process mining algorithms, in any of these branches, work under the assumption that behaviour related to the execution of the underlying process is stored correctly within the event log. Moreover, completeness of the behaviour, 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 noise, i.e., behaviour that is/should not be part of the process. Furthermore, it often contains data related to infrequent behaviour, i.e., behaviour that is rather rare due to the handling of exceptional cases. The presence of such behaviour, which we subsequently refer to as *outlier*, makes most process mining algorithms return complex, incomprehensible or even inaccurate results. To reduce these negative effects, process mining projects often comprise of a pre-processing phase in which one tries to detect and remove traces that contain such undesired behaviour. This cleaning phase is usually performed manually and is therefore rather costly and time consuming.

Despite the negative impacts of the presence of noisy, incomplete and infrequent behaviour, little research has been done towards automated data cleansing techniques that improve process mining results. Recently, research has been performed aiming at *filtering* out traces that contain outlier behaviour from an event log [3, 4]. Even though both techniques show improvements in process mining results, in particular w.r.t. process discovery, only a little fragment of outlier behaviour within a trace of event data leads to ignoring the trace as a whole. This potentially leads to a distortion of the general distribution of common behaviour of the process, yielding potentially inaccurate or even wrong process mining results.

Therefore, we propose a general purpose *event log repairing technique* that, given event potentially containing outlier behaviour, identifies and modifies such behaviour in order to obtain a more reliable input for all possible process mining algorithms. In particular, we use a probabilistic method to detect outlier behaviour according to the *context* of a process i.e., fragments of activity sequences that occur before and after the potential outlier behaviour. After outlier identification, the corresponding behaviour is replaced with another fragment of behaviour that is more probable to occur according to the context in which the outlier behaviour occurs.

Using the `ProM` [5] based extension of `RapidMiner`, i.e., `RapidProM` [6], we study the effectiveness of our approach, using both synthetic and real event data. The obtained results show that our approach adequately identifies and repairs outlier behaviour and as a consequence increases the overall quality of process model discovery results. Additionally, we show that our method achieves better results compared to one of the best performing existing filtering techniques in the process mining domain.

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

2 Motivation

Real event logs often contain noise/anomalous behaviour. The presence of such behaviour causes many problems for process mining algorithms. In particular, most process discovery algorithms incorporate all behaviour in event logs as much as possible. As a result, most outlier behaviour is incorporated as well which decreases the overall accuracy of the discovered model. Therefore, it is essential to accurately pre-process

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

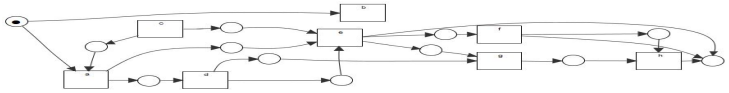
Variant	Frequency
$\langle a, b, c, d, e, f, h\rangle$	2
$\langle a, b, d, c, e, f, h\rangle$	1
$\langle a, b, c, d, e, g, h\rangle$	1
$\langle a, b, d, c, e, g\rangle$	1
$\langle b, d, c, e, g, h\rangle$	1
$\langle a, b, c, d, e\rangle$	1
$\langle a, b, c, d, g, h\rangle$	1
$\langle a, b, b, c, d, e, f, h\rangle$	1
$\langle a, b, c, d, g, h\rangle$	1
$\langle a, b, d, c, a, e, g, f, h\rangle$	1

event data. In fact, according to the process mining manifesto [7], cleaning event logs is one of the main challenges in the *process mining* field.

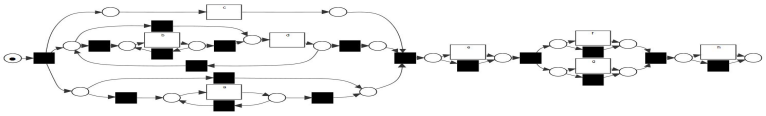
It is possible to define *outlier behaviour* in a variety of ways. For example, in [2], *infrequent behaviour* is considered as outlier behaviour. In turn, infrequent behaviour relates to rare behaviour that is supposed to be part of the process, yet severely hampers the feasibility of process mining results. In practice, behaviour that is not part of the process, i.e. caused by logging exceptions, faulty execution of the process, is infrequent by its sheer nature. So, in this paper we define both infrequent behaviour that is part of the process and (infrequent) faulty execution behaviour, i.e. noise, as outlier behaviour.

An example event log with some outlier behaviour is shown in Table 1. In this event log there are 74 events belong to 11 traces. Except for the first variant (unique behaviour of a process instance) each variant has just one corresponding trace, note that this is customary in some application domains, e.g. medical treatment process [8]. The first three traces contain no outlier behaviour. However, the next seven variants have different types of outlier behaviour. For example, in the fourth and fifth rows the activities "*h*" and "*a*" are missing. Some process discovery algorithms like the Alpha miner [9] are sensitive to such outlier behaviour and yield inferior process discovery results when applied directly to such event logs. Other process discovery algorithms like the Inductive Miner [10], have embedded filtering mechanisms to deal with some types of outliers. The results of applying various process discovery algorithms on this event log are shown in Figure 1. If we apply filtering method of [4] on this event log, variants $1-5$ are retained. A resulting process model using these traces combined with the Inductive Miner is shown in Figure 1e. As it is shown in this figure, all mentioned process discovery algorithms have problem to discover an accurate model from the given event log. However, if we first repair the event log and then apply the Alpha miner (or any other mentioned process discovery methods), we obtain an accurate process model, i.e. as presented in Figure 1f. It is because there is no outlier behaviour in the repaired event log and the resulted process model is more accurate and understandable.

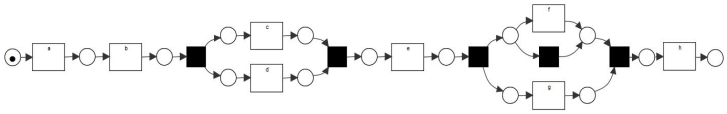
So, it seems that we need an approach to overcome outlier behaviour in event data. A naive approach to solve data quality related issues is to remove traces that seem to contain outlier behaviour [3, 4]. However, for many businesses, all process instances in an event log are valuable and ignoring them potentially jeopardizes the trustworthiness of the analysis performed. For example, in a patient treatment in a hospital, recorded over several years, it is undesirable to remove all process related records of a patient just



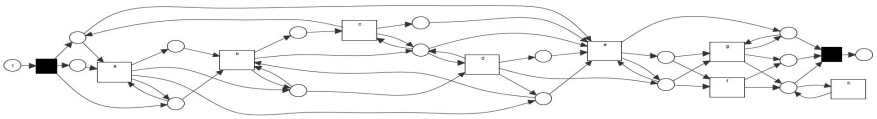
(a) Result of the Alpha Miner on event log of Table 1



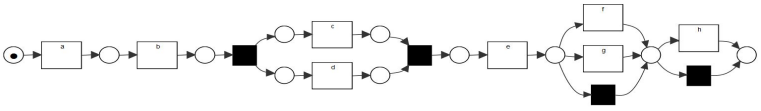
(b) Result of the Inductive Miner on event log of Table 1



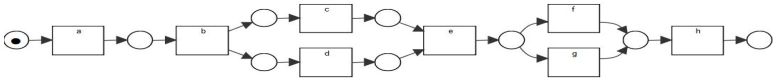
(c) Result of the Inductive Miner with integrated filtering (IMi) on event log of Table 1



(d) Result of the ILP Miner on event log of Table 1



(e) Result of the Inductive Miner on the filtered event log by [4]



(f) Applying Alpha miner on the even log of Table 1 after repairing outlier behaviour in it.

Fig. 1: Resulting process models of applying different process discovery techniques (with and without filtering/repair) on the event log that is presented in Table 1

because there exists a small portion of wrongly logged behaviour. In such scenarios, after detecting outlier, it is more desirable to repair that behaviour. Note that, if there is no noise in an event log, by repairing infrequent behaviour in it and removing too detailed patterns, we are able to alter infrequent behaviour into more frequent behaviour which allows us to discover more general views on the process. Therefore, by repairing, rather than removing outlier behaviour we believe that quality of discovered process models improves.

3 Related Work

Some process mining algorithms are designed to/have been extended to be able to handle outliers as well [11–14]. However, these filtering techniques are tailored towards the

internal working of the corresponding algorithms and cannot be used for general purpose event log cleansing. Additionally, they typically focus on a specific type of noise, e.g. incompleteness.

Most of filtering techniques present in commercial tools are based on just the frequency of activities and variants. However, the presence of parallelism and loops often hampers the applicability of such filtering techniques. There are also some basic filtering plug-ins developed in ProM [5] based on activity frequencies and users inputs.

Outlier detection for general temporal data is addressed in some research, e.g. in [15] a survey on different methods of detecting outliers in sequential data is presented. Also, there are some related techniques that specifically proposed for process mining domain. In [16, 17] the authors propose filtering techniques that use additional information such as training event data or a reference process model. In reality, providing a sufficiently complete set of training traces or having a reference process model is impractical. Recently, some general purpose filtering techniques are proposed in the process mining domain. In [3] by constructing an Anomaly Free Automaton (AFA) based on the whole event log and a given threshold, all non-fitting behaviour, w.r.t. the AFA, is removed from the event log. In [4], we propose a filtering method that detects outliers based on conditional probabilities of subsequences and their possible following activities. In [18],an adjustable on-line filtering method is proposed that detect outlier behaviour for streaming event logs that also works based on conditional probabilities.

All aforementioned methods, after detecting outlier behaviour, try to remove such behaviour. As motivated in Section 2, modifying outlier behaviour or repairing it is more valuable than just removing it. There is some research [19, 20] that tries to repair process models based on event logs. Moreover, [21] uses process model to repair event logs. However, as we aim to design a general purpose repairing method, we assume there does not exist a process model as an input of our proposed algorithm.

4 Repairing Outliers in Event Data

In this section we present our outlier repair method. Prior to this, we briefly introduce basic process mining terminology and notations that ease readability of the paper.

4.1 Terminology and Notation

Given a set X, a multiset M over X is a function $M\colon X \to \mathbb{N}_{\geq 0}$, i.e. it allows certain elements of X to appear multiple times. We write a multiset as $M = [e_1^{k_1}, e_2^{k_2}, ..., e_n^{k_n}]$, where for $1 \leq i \leq n$ we have $M(e_i) = k_i$ with $k_i \in \mathbb{N}_{>0}$. If $k_i = 1$, we omit its superscript, and if for some $e \in X$ we have $M(e) = 0$, we omit it from the multiset notation. Also, $M = [\,]$ denotes an empty multiset, i.e. $\forall e \in X,\ M(e) = 0$. We let $\overline{M} = \{e \in X \mid M(e) > 0\}$, i.e. $\overline{M} \subseteq X$. The set of all possible multisets over a set X is written as $\mathcal{M}(X)$.

Let X^* denote the set of all possible sequences over a set X. A finite sequence σ of length n over X is a function $\sigma\colon \{1, 2, ..., n\} \to X$, 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

Table 2: Fragment of a fictional event log (each line corresponds to an event).

Case-id	Activity	Resource	Time-stamp
...	...	...	...
1	*register request (a)*	*Sara*	*2017-04-08:08.10*
1	*examine thoroughly (b)*	*Ali*	*2017-04-08:09.17*
2	*register request (a)*	*Sara*	*2017-04-08:10.14*
1	*check resources (c)*	*William*	*2017-04-08:10.23*
1	*check ticket (d)*	*William*	*2017-04-08:10.53*
2	*check resources (b)*	*Ali*	*2017-04-08:11.13*
1	*Send to manager(e)*	*Ava*	*2017-04-08:13.09*
1	*accept request (f)*	*Fatima*	*2017-04-08:16.05*
1	*mail decision(h)*	*Anna*	*2017-04-08:16.18*
...	...	...	...

ϵ. Concatenation of sequences σ and σ' is written as $\sigma \cdot \sigma'$. We let function $hd\colon X^* \times \mathbb{N}_{\geq 0} \nrightarrow X^*$, 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, $tl\colon X^* \times \mathbb{N}_{\geq 0} \nrightarrow X^*$ 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 $\exists \sigma_1, \sigma_2 \in X^* (\sigma = \sigma_1 \cdot \sigma' \cdot \sigma_2)$. Let $\sigma, \sigma' \in X^*$. We define the frequency of occurrence of σ' in σ by $freq\colon X^* \times X^* \to \mathbb{N}_{\geq 0}$ where $freq(\sigma', \sigma) = |\{1 \leq i \leq |\sigma| \mid \sigma'_1 = \sigma_i, ..., \sigma'_{|\sigma'|} = \sigma_{i+|\sigma'|}\}|$. Given an example event log that presented in Table 1, $freq(\langle b \rangle, \langle a, b, b, c, d, e, f, h \rangle) = 2$ and $freq(\langle b, d \rangle, \langle a, b, d, c, e, g \rangle) = 1$, etc.

Event logs describe sequences of executed business process activities, typically in context of some cases (or process instances), e.g., a customer or an order-id. 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 [2], is presented in Table 2.

Consider the events related to *Case-id* value 1. Sara *registers a request*, after which Ali *examines it thoroughly*. William *checks the ticket* and *checks resources*. Ava *sends the request to manager* and Fatima *accept the request*. Finally, Anna *emails the decision* to the client. The 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 formally define event logs as a multiset of sequences of activities (e.g., Table 1).

Definition 1 (Event Log). *Let $\mathcal{A}$ be a set of activities. An event log is a multiset of sequences over $\mathcal{A}$, i.e. $L \in \mathcal{M}(\mathcal{A}^*)$.*

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.

4.2 Repairing Event Logs Using Control-Flow Oriented Contexts

Here, we present our methodology to identify and repair outlier behaviour in an event log. We first present our repair method by means of an example after which we for-

malize and discuss it in more detail. We first present two central control-flow oriented concepts which form the basis of the repair method. Given a trace, a context is defined as the surrounding behaviour of a certain sequence of activities. For example, in trace $\langle a, b, c, d, e, f, h\rangle$, $\langle a, b\rangle$ and $\langle e\rangle$ are surrounding $\langle c, d\rangle$, hence, the pair $(\langle a, b\rangle, \langle e\rangle)$ is a context of $\langle c, d\rangle$.

Definition 2 (Context, Covering). *Let $\sigma, \sigma' \in X^*$. We define the context of σ' w.r.t. σ as a function $con : X^* \times X^* \times \mathbb{N}_{\geq 0} \times \mathbb{N}_{\geq 0} \to \mathcal{P}(X^* \times X^*)$ where $con(\sigma', \sigma, l, r) = \{(\sigma_1, \sigma_2) \in X^* \times X^* \mid \sigma_1 \cdot \sigma' \cdot \sigma_2 \in \sigma \wedge |\sigma_1| = l \wedge |\sigma_2| = r\}$.*

Furthermore, let $\sigma'_1, \sigma'_2 \in X^*$, $cov : X^* \times X^* \times X^* : \to \mathcal{P}(X^*)$ is a function that returns all subsequences in a trace that occur within a given context, i.e. $cov(\sigma, \sigma'_1, \sigma'_2) = \{\sigma' \in X^* | (\sigma'_1, \sigma'_2) \in con(\sigma', \sigma, |\sigma'_1|, |\sigma'_2|)\}$.

A context describes, given a subsequence, its two neighbouring subsequences of length l and r respectively, i.e. σ'_1 is the left neighbour with length l and σ'_2 is the right neighbor with length r. For example, if $\sigma = \langle a, b, c, d, e, f, h\rangle$, we have $con(\langle c\rangle, \sigma, 1, 2) = (\langle b\rangle, \langle d, e\rangle)$. Note that l and r may differ, and, may even have value equal to 0 which implies a neighboring subsequence ϵ. Furthermore, $cov(\sigma, \langle a, b\rangle, \langle e, f\rangle) = \{\langle c, d\rangle\}$ and $cov(\sigma, \langle b\rangle, \langle d\rangle) = \{\langle c\rangle\}$.

We aim to detect and replace outlier behaviour based on the occurrence probability of a subsequence among a specific context. If this probability is lower than a given threshold, we consider that behaviour as outlier. To obtain these probabilities we define *covering probability* as follows.

Definition 3 (Covering Probability). *Given $\sigma', \sigma'_1, \sigma'_2 \in X^*$, maximum subsequences' length K and a multiset of sequences $L \in \mathcal{M}(X^*)$. We define $CP\colon X^* \times X^* \times X^* \times \mathbb{N}_{>0} \times \mathcal{M}(X^*) \to [0, 1]$ as the empirical conditional probability of σ' being covered by σ_1 and σ_2 in event log L, i.e.*

$$CP(\sigma', \sigma'_1, \sigma'_2, K, L) = \begin{cases} \frac{\sum_{\sigma \in L} \left(L(\sigma) freq(\sigma'_1 \cdot \sigma' \cdot \sigma'_2, \sigma)\right)}{\sum_{\sigma \in L} \left(L(\sigma) \sum_{|\sigma''| \leq K} freq(\sigma'_1 \cdot \sigma'' \cdot \sigma'_2, \sigma)\right)} & \text{if } \exists \sigma \in L (cov(\sigma', \sigma'_1, \sigma'_2) \neq \emptyset) \\ 0 & \text{otherwise} \end{cases}$$

The numerator computes how many times σ' is covered by the context (σ'_1, σ'_2) in whole event log. The denominator computes how many times context (σ'_1, σ'_2) covers different substrings with length lower or equal than K. Clearly, the resulting value is a real number in $[0, 1]$. A higher value implies that it is more probable that subsequence σ' occurs among context (σ'_1, σ'_2). So, $CP(\sigma', \sigma'_1, \sigma'_2, 1, L) = 1$, indicates that if context (σ'_1, σ'_2) occurs, subsequence σ' always happens among it. According to the event log that is presented in Table 1, $CP(\epsilon, \langle b\rangle, \langle c\rangle, 1, L) = \frac{7}{12}$ and $CP(\langle b\rangle, \langle b\rangle, \langle c\rangle, 1, L) = \frac{1}{12}$.

The main idea of our approach is that if the covering probability of a subsequence among a *significant context* is lower than the given threshold, the covered subsequence is considered as outlier. A significant context is simply a context that occurs significantly often, where the significance of occurrence is a user-defined threshold. Subsequently, we substitute the outlier subsequence with another subsequence that has a higher covering probability among the given context.

Significant contexts	Frequency	Probable subsequences
(∅, ⟨a⟩)	98	∅
(∅, ⟨b⟩)	70	⟨a⟩
(∅, ⟨c⟩)	30	⟨a⟩
(⟨a⟩, ⟨b⟩)	101	∅, ⟨c⟩
(⟨a⟩, ⟨c⟩)	95	∅, ⟨b⟩
(⟨b⟩, ⟨c⟩)	67	∅
(⟨b⟩, ∅)	103	∅, ⟨c⟩
(⟨c⟩, ⟨b⟩)	35	∅
(⟨c⟩, ∅)	97	∅, ⟨b⟩

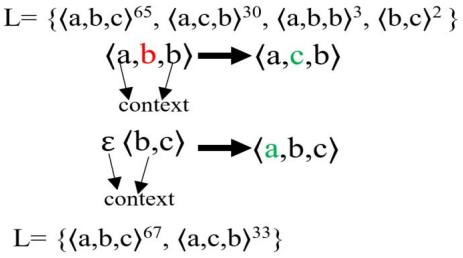


Fig. 2: A simple example of repairing an event log L with our proposed method. Significant contexts and their probable subsequences are shown in the table for $K = 1$.

In context of our example, in the trace $\langle a, b, b, c, d, e, f, h\rangle$, if we consider $(\langle b\rangle, \langle c\rangle)$ as a frequent context, by replacing subsequence $\langle b\rangle$ with ϵ, which is more probable to happen among this context, we obtain $\langle a, b, c, d, e, f, h\rangle$, which is not an outlier any more. As another example, consider trace $\sigma = \langle a, d, b, d, c, f, h\rangle$. By having a significant context $(\langle a\rangle, \langle b\rangle)$, according to the whole event log, we expect that ϵ occurs among it rather than $\langle d\rangle$. Therefore, by this replacement we will have $\langle a, b, d, c, f, h\rangle$. Similarly, for $\langle a, b, c, d, g, h\rangle$ by considering context (d, g) and considering covering probabilities, we will replace ϵ by $\langle e\rangle$ and we obtain $\langle a, b, c, d, e, g, h\rangle$ which is not outlier any more.

The user decides which contexts are significant by setting a corresponding significance threshold. Contexts with a frequency of at least the number of traces in the event log times the given threshold are considered as significant contexts. Also, the maximum length of covered subsequence (K) and context's subsequences (C_L) respectively are specified by the user. Note that C_L describes two values, i.e. a maximal length for σ'_1 and σ'_2.

The input of our proposed method is an event log L, a context frequency threshold T_C, a context's subsequence lengths $C_L(l, r)$ and an upper bound for length of covered subsequence K. At first, all traces are scanned to compute covering probabilities of all contexts and their possible covered subsequences. Next, for each trace and each subsequence in it (with length $\leq K$), we check its context frequency and covering probability (according to T_C). If the context is significant and the covering probability is low, we replace the subsequence with a better one according to that context. Otherwise, if a context is insignificant it is not able to be used for repairing outlier behaviour.

A simple visual example of how the proposed method works is given in Figure 2.

In this example, we consider $K = 1$ as a maximum subsequence length and also context lengths are equal to 1 to repair an event log with 100 traces. First, by scanning the event log, significant contexts and their probable subsequences are specified. If the corresponding context is significant and the subsequence is not probable for that context, we detect a noisy/infrequent behaviour. For example, the occurrence of $\langle b\rangle$ is improbable among significant context $(\langle a\rangle, \langle b\rangle)$.

After detecting outlier behaviour, we try to replace improbable subsequences with more probable ones. For substitution, we are searching for a subsequence with a length as close as possible to the outlier subsequence. Among all candidate subsequences we are interested in the subsequence with the highest probability. For example, if the outlier subsequence has length 2 we first search for a subsequence with the same length. If there is not any significant subsequence with length 2 for that context, we try to find a subsequence σ'' with length 1 or 3. Then, among the candidate subsequences we choose one that has the highest probability among that context. According to Figure 2 there are two possible subsequences to substitute with $\langle b \rangle$, i.e. ϵ and $\langle c \rangle$. Because the length of $\langle c \rangle$ is similar to the outlier subsequence we choose it and the trace is changed into $\langle a, c, b \rangle$. For the other outlier trace in this example, among the context $(\epsilon, \langle b \rangle)$, it is more probable that $\langle a \rangle$ occurs. So, by replacing ϵ with $\langle a \rangle$, it changes to $\langle a, b, c \rangle$.

To avoid having infinitive loops, after each replacement, the starting point of the next scanned subsequence will be the first point of the right subsequence of the previous context. For example, after replacement of $\langle a \rangle$ with ϵ and having $\langle a, b, c \rangle$, we will not check $\langle a \rangle$ again as a subsequence as we consider $\langle b \rangle$ as the next subsequence.

5 Evaluation

To evaluate our proposed method we conducted several experiments with both artificial and real event data. To simplify the evaluation, in all the experiments we just consider contexts' subsequences length equal to 1.

To apply the proposed repairing method, we implemented the *Repair Log* plug-in (*RL*) in the `ProM` framework[1]. The plugin takes an event log as an input and outputs a repaired event log. Also, the user is able to specify threshold T_C, the maximum subsequence length K, and the length of left and right sequences of context C_L.

In addition, 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* plugin to `RapidProM`. `RapidProM` is an extension of `RapidMiner` that combines scientific workflows [22] with a range of (`ProM`-based) process mining algorithms.

To evaluate discovered process models, we use fitness and precision. Fitness computes how much behaviour in the event log is also described by the process model. However, precision measures how much of behaviour described by the model is also presented in the event log. Low precision means that the process model allows for much more behaviour compared to the event log. Note that, there is a trade off between these measures [23]. Sometimes, putting aside a small amount of behaviour causes a slight decrease in fitness, whereas precision increases much more. Therefore, to evaluate discovered process models, we use the F-Measures metric that combines fitness and precision: $\frac{2 \times Precision \times Fitness}{Precision + Fitness}$. Note that, in many applications, fitness has more importance. We therefore also use the notion of conditional precession, in which we only consider precision values of those process models that have $fitness \geq 0.95$. Furthermore, as shown in [4], *Matrix Filter* has a good performance on event logs with outlier

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

behaviour. Hence, we compare the results of the proposed repairing method with this filtering method.

Table 3: Details of real event logs that are used in the experiment

Event Log	Activities Count	Traces Count	Events Count	Variants Counts
BPIC_2017_Offer	8	42995	193849	16
BPIC_2013	13	7554	65533	2278
BPIC_2012_Application	10	13087	60849	17
BPIC_2012_Workflow	6	9658	72413	2263
BPIC_2012_Offer	7	5015	31244	168
$Road\_Fines$	11	150370	561470	231
$Hospital\_Billing$	18	100000	451359	1020
$Sepsis\_Cases$	16	1050	15224	846
$Credit$	15	10035	150525	1

In the first experiment, we try to investigate the effect of using the proposed method on real event logs. Basic information of these event logs is presented in Table 3. We also add different percentages of noise to these event logs. As noise we consider addition of random activities at random positions in traces, random removal of activities, and swapping of activities within traces. For example, we added 5% of all these three mentioned types of noise to $Road\_Fines$ and refer to it as $Road\_Fines\_05$. Note that, in all experiments, we just used the modified (filtered or repaired) event logs for discovering process models. For conformance checking and quality assessment of discovered process models we used original event logs (without added noise).

We try to discover the best process models according to the F-Measure for these event logs using four different methods. In the first method, we apply *IMi* [11] with 51 different noise filtering thresholds (that is embedded in the Inductive Miner) from 0 to 1 and we call it *N&IMi* (*N* means that event logs are not filtered beforehand). In the second method, we apply the basic Inductive Miner (without its embedded filtering) on an event log that is previously filtered with *Matrix Filter* method and call it as *M&IM* (*M* means event logs are filtered with *Matrix Filter* beforehand). We also apply the basic Inductive Miner on event logs that are repaired beforehand and name it as *R&IM* (*R* means event logs are repaired with the proposed method beforehand). In the last method, the Inductive Miner with four different noise filtering thresholds (from 0.1 to 0.4) is applied on repaired event logs (*R&IMi*).

The results of applying these methods to aforementioned event logs and their best F-Measure values are given in Figure 3 and Figure 4. These figures show that when there is outlier behaviour in the event log, just using the Inductive Miner (*N&IMi*) does not yield a proper process model. However, in some event logs that do not contain outlier behaviour, the Inductive Miner discovers a process model with a suitable F-Measure. It is also notable that for most event logs, *R&IM* outperforms *M&IM* and allows us to obtain better results. But, for the $Hospital\_Billing$ event log, the result of *M&IM* is slightly better in the previous experiments. This is because there are lots of variants in this event log, but 94% of traces are placed in just 1% of variants and in this type of event logs, filtering works perfect. The best results are achieved when we apply Inductive Miner on cleaned event logs or *R&IMi*. Moreover, in most of cases, *M&IM* and the Inductive Miner (here *N&IMi*) are achieving their best possible process

model according to F-Measure, by sacrificing a lot in term of fitness. In Figure 5 and Figure 6, we see the result of different methods according to their conditional precision. According to these figures, the *R&IM* method helps the Inductive Miner to discover process models with high precision without sacrificing the fitness. However, we see that for event logs with lots of outlier behaviour the Inductive Miner does not able to discover a process model with high fitness and precision at the same time.

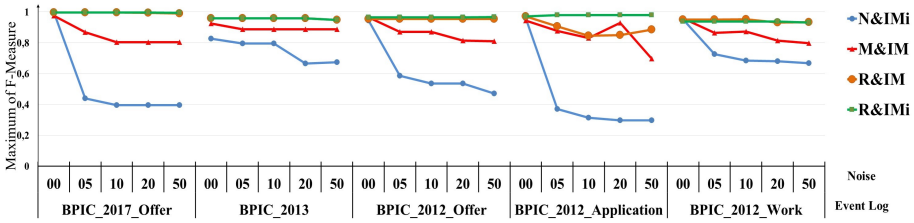


Fig. 3: Best F-Measure of applying different methods on *BPIC* event logs.

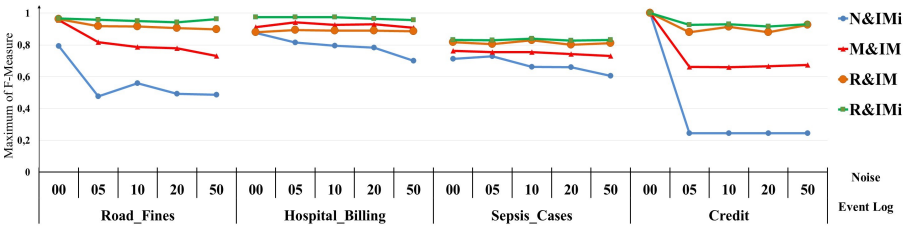


Fig. 4: Best F-Measure of applying different methods on other real event logs.

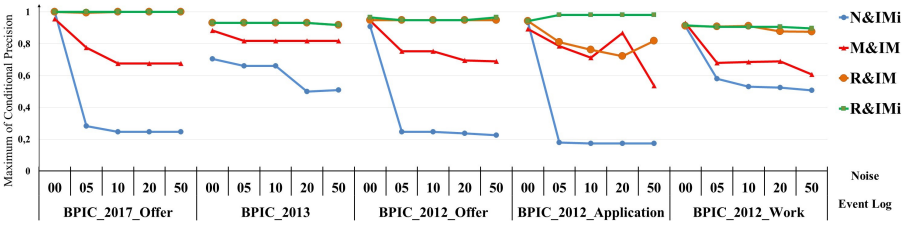


Fig. 5: Best conditional precision of applying different methods on *BPIC* event logs.

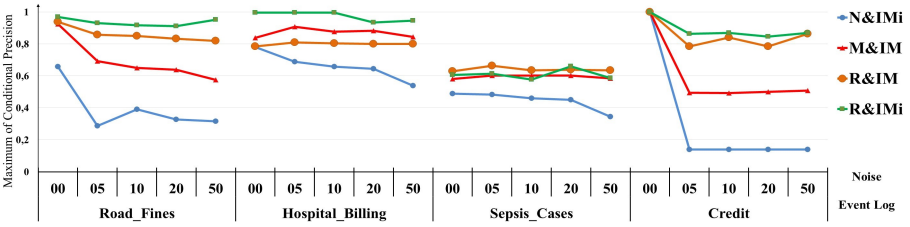
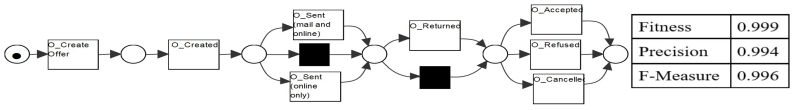
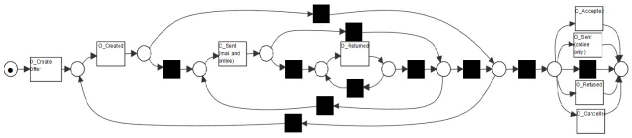


Fig. 6: Best conditional precision of applying different methods on other real event logs.



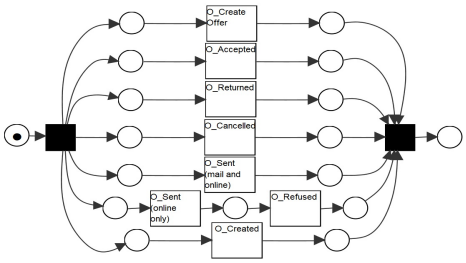
Fitness	0.999
Precision	0.994
F-Measure	0.996

(a) Results of Applying *R&IM*



Fitness	0.989
Precision	0.776
F-Measure	0.870

(b) Results of applying *M&IM*



Fitness	1.000
Precision	0.282
F-Measure	0.440

(c) Results of applying *N&IMi*

Fig. 7: The best discovered process models on $BPIC\_2017\_Offer\_05$ using *RL*, *MF* and *NF* methods.

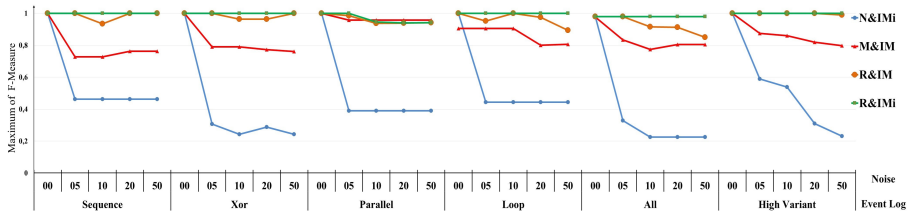


Fig. 8: Best F-Measure of applying different methods on synthetic event logs.

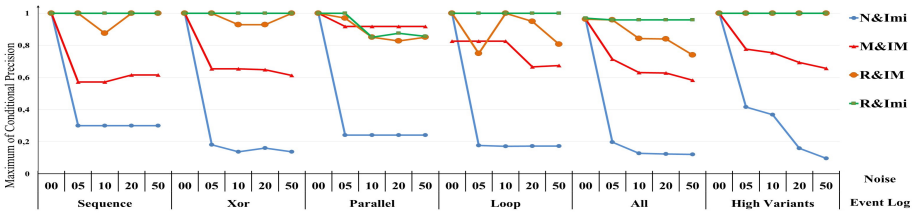


Fig. 9: Best conditional precision of applying different methods on synthetic event logs.

Note that for some event logs like the $Sepsis\_Cases$ neither *M&IM* nor *R&IM* allow us to discover an outstanding process model, i.e, having high conditional precision. The reason is for this is related to the fact that the discovered process model of this

event log is not precise and there is some behaviour that possible to happen anywhere during execution of the process. We have shown the best discovered process model for this event log to a business expert which indicated that the process model is acceptable (but not perfect).

In Figure 7, the best process model of each methods for the $BPIC\_2017\_Offer\_05$ are shown. It is obvious that Figure 7a that is discovered on the repaired event log is the best process model among them.

Note that, in the previous experiments, the original event log that is used to compute F-Measures (our ground truth) potentially also contains noisy and infrequent behaviour. Therefore, in another experiment, we create six artificial process models with different behaviour. These process models consequently describe different types of behaviour, i.e. one model just contains sequence constructs (*Sequence*), one contains sequence and choice constructs (*Xor*), one contains sequence and parallel constructs (*Parallel*), one contains sequence and loop constructs (*Loop*), one contains all possible behavioural constructs (*All*) and we use a process model with locally structured subprocesses that has lots of variants (*High Variants*). Based on these reference process models, we generated six event logs that each contains 5000 traces. Similar to the previous experiment, 5%,10%,20% and 50% of noise inserted to these original event logs. Note that, an event log with 10% inserted noise not necessary has 90% original traces. Similarly, we apply *N&IMi*, *M&IM*, *R&IM* and *R&IMi* to these event logs. Results of this experiment are given in Figure 8 and Figure 9.

As shown in these figures, for all event logs if we insert some noisy behaviour (even just 5%), the Inductive Miner has problems to discover a process model with a high F-Measure. This means the embedded noise filtering method mechanism in this algorithm is not able to deal with all types of outlier. Furthermore, *Matrix Filter* always works better than the embedded noise filtering method in the Inductive Miner. Compared to these filtering methods, the proposed repaired method performs better especially when there probability of noise is low. Only for the *Parallel* event log, results of applying *M&IM* slightly outperforms our proposed method. Like the previous experiment, the best results are achieved when we apply the Inductive Miner with its filtering method to the repaired event logs (*R&IMi*). Except for the *Parallel* event logs, for all others this method finds the underlying process models, i.e. almost equal to the reference process models. For the *High Variants* event log that has local structured subprocesses, our proposed repair method works perfectly and accurately detects and repairs outlier behaviour. The reason for this is related ot the fact that our method is able to detect and repair outlier behaviour locally.

Table 4: The percentage of remaining traces in event logs for the best process model

Noise Percentage / Event Log	**0**	**5**	**10**	**20**	**50**
Sequence	1	95	90	81	52
Xor	1	95	90	42	52
Parallel	1	52	48	41	23
Loop	1	95	90	80	0,51
ALL	83	95	90	9	5
High Variants	1	95	90	81	20

Finally, note that filtering methods achieve the best results by removing a lot of behaviour in event logs. The percentages of remaining traces in each event log for the best model in *M&IM* are given in Table 4. In some cases the best process model is discovered just with 5% of the traces. This means that we need to remove a lot of behaviour from the event log. However, in the repair method all the traces remain in the event log but they may be modified.

Note, for all methods we use a grid search on different parameters and show the best results obtained. However, in reality like other state-of-the-art process mining specific data cleansing methods, adjusting these thresholds is a challenging task for users.

6 Conclusion

Process mining provides insights into the actual execution of business processes, by exploiting available event data. Unfortunately, most process mining algorithms are designed to work under the assumption that the input data is outlier free. However, real event logs contain outlier (noise and infrequent) behaviour that typically leads to inaccurate/unusable process mining results. Detecting such behaviour 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 takes an event log and returns a repaired event log. It uses the occurrence frequency of a control-flow oriented context pattern and the probabilities of different subsequences appearing in middle of it to detect outlier behaviour. If such probability be lower than a given threshold, the subsequence is substituted with a more probable one according to the context.

To evaluate the proposed repairing method we developed a plug-in in the ProM platform and also offer it through 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 the proposed method on synthetic event logs. The results indicate that the proposed repair approach is able to detect and modify outlier behaviour and consequently is able to help process discovery algorithms to return models that better balance between different behavioural quality measures. Furthermore, using these experiments we show that our repair method outperforms one of the best state-of-the-art process mining filtering techniques as well as the Inductive Miner algorithm with its embedded filtering mechanism.

As future work, we want to evaluate the proposed method on other process mining results and also other process discovery algorithms. 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.

References

1. van der Aalst, W.M.P.: Using Process Mining to Bridge the Gap between BI and BPM. IEEE Computer **44**(12) (2011) 77–80
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer Berlin Heidelberg (2016)

3. Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: Filtering Out Infrequent Behavior from Business Process Event Logs. IEEE Trans. Knowl. Data Eng. **29**(2) (2017) 300–314
4. Fani Sani, M., van Zelst, S.J., van der Aalst, W.M.: Improving Process Discovery Results by Filtering Outliers Using Conditional Behavioural Probabilities. In: International Conference on Business Process Management, Springer (2017) 216–229
5. van der Aalst, W., van Dongen, B.F., Günther, C.W., Rozinat, A., Verbeek, E., Weijters, T.: ProM: The Process Mining Toolkit. BPM (Demos) **489**(31) (2009)
6. van der Aalst, W.M.P., Bolt, A., van Zelst, S.J.: RapidProM: Mine Your Processes and Not Just Your Data. CoRR **abs/1703.03740** (2017)
7. van der Aalst, W., et al.: Process Mining Manifesto. In: International Conference on Business Process Management, Springer (2011) 169–194
8. Rebuge, Á., Ferreira, D.R.: Business Process Analysis in Healthcare Environments: A Methodology Based on Process Mining. Information systems **37**(2) (2012) 99–116
9. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow Mining: Discovering Process Models From Event Logs. IEEE Trans. Knowl. Data Eng. **16**(9) (2004) 1128–1142
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Application and Theory of Petri Nets and Concurrency. Springer Berlin Heidelberg (2013) 311–329
11. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In: Business Process Management Workshops. Springer International Publishing (2014) 66–78
12. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering Workflow Nets Using Integer Linear Programming. Computing (Nov 2017)
13. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM). In: CIDM. (2011)
14. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining –Adaptive Process Simplification Based on Multi-perspective Metrics. In: Lecture Notes in Computer Science. Springer Berlin Heidelberg (2007) 328–343
15. Chandola, V., Banerjee, A., Kumar, V.: Anomaly Detection for Discrete Sequences: A Survey. IEEE Transactions on Knowledge and Data Engineering **24**(5) (2012) 823–839
16. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning Structured Event Logs: A Graph Repair Approach. In: ICDE 2015. (2015) 30–41
17. Cheng, H.J., Kumar, A.: Process Mining on Noisy Logs —Can Log Sanitization Help to Improve Performance? Decision Support Systems **79** (2015) 138–149
18. van Zelst, S.J., Fani Sani, M., Ostovar, A., Conforti, R., La Rosa, M.: Filtering Spurious Events from Event Streams of Business Processes. In: Proceedings of the CAISE. (2018)
19. Fahland, D., van der Aalst, W.: Model Repair—Aligning Process Models to Reality. Information Systems **47** (2015) 220–243
20. Armas-Cervantes, A., van Beest, N., La Rosa, M., Dumas, M., Raboczi, S.: Incremental and Interactive Business Process Model Repair in Apromore. Proceedings of the BPM Demos. CRC Press (2017, to appear) (2017)
21. Rogge-Solti, A., Mans, R.S., van der Aalst, W.M.P., Weske, M.: Improving Documentation by Repairing Event Logs. In: IFIP Working Conference on The Practice of Enterprise Modeling, Springer (2013) 129–144
22. Bolt, A., de Leoni, M., van der Aalst, W.M.P.: Scientific Workflows for Process Mining: Building Blocks, Scenarios, and Implementation. STTT **18**(6) (2016) 607–628
23. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A Robust F-measure for Evaluating Discovered Process Models. In: Proceedings of the CIDM, pages = 148–155, year = 2011,