Applying Sequence Mining for Outlier Detection in Process Mining

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

[1] Process and Data Science Chair, RWTH Aachen University, 52056 Aachen, Germany
{fanisani,wvdaalst}@pads.rwth-aachen.de
[2] Fraunhofer FIT, Birlinghoven Castle, Sankt Augustin, Germany
s.j.v.zelst@pads.rwth-aachen.de

Abstract. One of the challenges in applying process mining algorithms on real event data, is the presence of outlier behavior. Such behaviour often leads to complex, incomprehensible, and, sometimes, even inaccurate process mining results. As a result, correct and/or important behaviour of the process may be concealed. In this paper, we exploit sequence mining techniques for the purpose of outlier detection in the process mining domain. Using the proposed approach, it is even possible to detect outliers in case of heavy parallelism and/or long-term dependencies between business process activities. Our method has been implemented in both the ProM- and the RapidProM framework. Using these implementations, we conducted a collection of experiments that show that we are able to detect and remove outlier behaviour in event data. Our evaluation clearly demonstrates that the proposed method accurately removes outlier behaviour and, indeed, improves process discovery results.

Keywords: Process mining · Sequence mining · Event log filtering
Event log preprocessing · Sequential rule mining · Outlier detection

1 Introduction

The main aim of process mining is to increase the overall knowledge of business processes. This is mainly achieved by (1) process discovery, i.e. discovering a descriptive model of the underlying process, (2) conformance checking, i.e. checking whether the execution of the process conforms to a reference model and (3) enhancement, i.e. the overall improvement of the view of the process, typically by enhancing a process model [1]. In each of these aspects, event data, stored during the execution of the process, is explicitly used to derive the corresponding results.

Many process mining algorithms assume that event data is stored correctly and completely describes the behavior of a process. However, real event data typically contains noisy and infrequent behaviour [2]. Usually, noise occurs rarely

H. Panetto et al. (Eds.): OTM 2018 Conferences, LNCS 11230, pp. 98–116, 2018.
https://doi.org/10.1007/978-3-030-02671-4_6

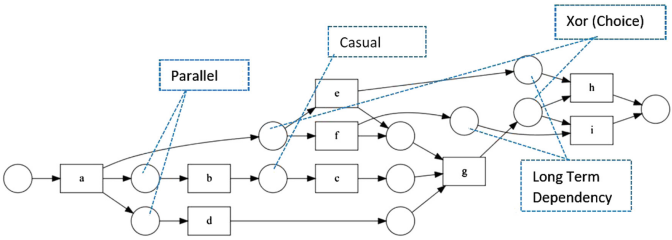


Fig. 1. An example of a process model with *Xor*, *parallel*, *long distance dependency* and *casual* behaviour.

in an event data, so, many researchers consider it as infrequent behaviour. Generally, noise relates to behaviour that does not conform to the process specification or its correct execution. However, infrequent behaviour refers to behaviour that is possible according to the process model, but, in exceptional cases of the process. Without having business knowledge, distinguishing between noise and infrequent behaviour is a challenging task. We, therefore, consider this as a separate research question and do not cover this in this paper. Here, we consider both noise and infrequent behaviour as ***outliers***.

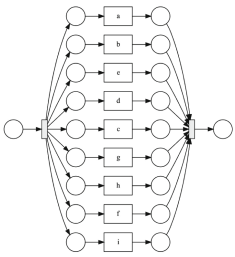
The presence of outlier behaviour makes many process mining algorithms, in particular, process discovery algorithms, result in complex, incomprehensible and even inaccurate results. Therefore, to reduce these negative effects, in process mining projects, often a preprocessing step is applied that aims to remove outlier behaviour and keep good behaviour. Such *preprocessing phase* increases the quality and comprehensiveness of possible future analyses. Usually, this step is done manually, which needs domain knowledge and it is costly and time-consuming.

In this paper, we focus on improving process discovery results by applying automated event data filtering, i.e., filtering the event log prior to apply any process discovery algorithm, without significant human interaction. Using sequential rules and patterns [3,4], the proposed filtering method is able to detect outlier behaviour even in event data with lot of concurrency, and long-term dependency behaviour. The presence of this type of patterns is shown to be hampering the applicability of automated existing general purpose filtering techniques [5,6]. By using the `ProM` [7] based extension of `RapidMiner`, i.e. `RapidProM` [8], we study the effectiveness of our approach, using synthetic and real event data. We show that our proposed filtering method more accurately detects outlier behaviour compared to existing event log filtering techniques in particular for event data with heavy parallel and long term dependences. Consequently, it increases more the overall quality of process discovery results.

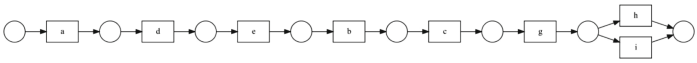
The remainder of this paper is structured as follows. Section 2 motivates the need for applying filtering methods on event logs. In Sect. 3, we discuss related work, then after explaining some preliminaries in Sect. 4, in Sect. 5, we describe our proposed method. Details of the evaluation and corresponding results are given in Sect. 6. Finally, Sect. 7 concludes the paper and presents directions for future work in this domain.

2 Motivation

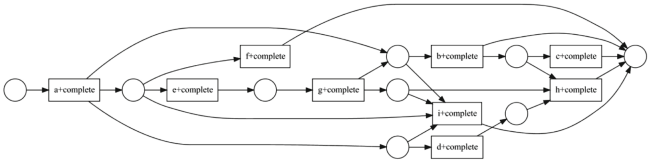
A process model allows us to (formally) describe what behaviour is (im)possible within a process. There are many different process modeling notations that allow us to model a process [1], e.g. in Fig. 1 we use a Petri net [9] to describe a process. This process model indicates that activities b, c, d, e, and f are in a parallel relation. However, between b and c there is a casual relation, i.e., activity c is only allowed to occur after the execution of activity b, but in between them other activities like d and e are allowed to be executed. Between e and f and also among h and i there are *Xor* (or exclusive choice) relations. Finally, the process model indicates that there are long term dependency relations between activities e and h and also between f and i. This means that, if in the middle of the process e happens, only activity h ends it.



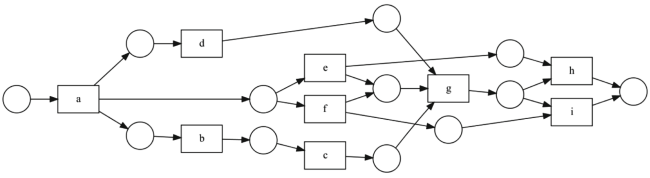
(a) Results of the Inductive Miner [10] on the raw event data.



(b) Result of the Alpha Miner [11] on the event log that filtered beforehand with *AFA* [6].



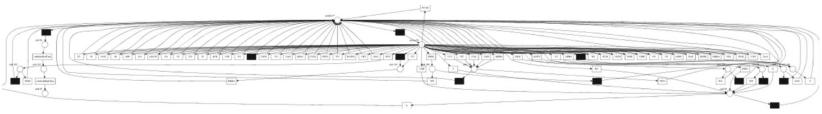
(c) Result of the Alpha Miner on the filtered event log using matrix filter [5] method (*SF*).



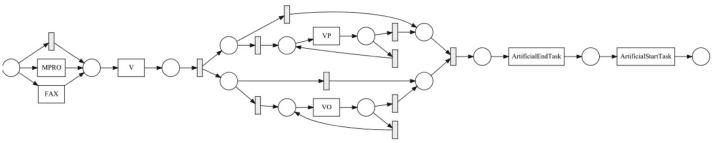
(d) Result of the Alpha Miner on the filtered event log using our proposed filtering method.

Fig. 2. Process models discovered on the noisy event log corresponding to Fig. 1.

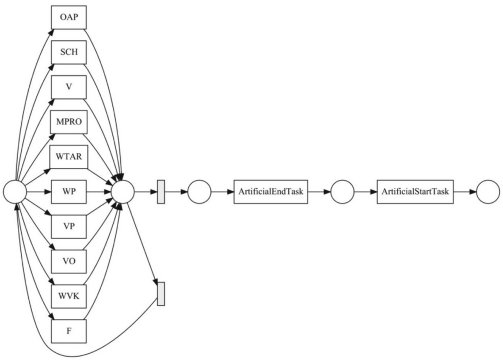
Process discovery algorithms aim to discover process models, i.e. models such as Fig. 1, on the basis of event data. The interpretability of such process model largely depends on the (graph-theoretical) complexity of such a model. However, often process discovery algorithms because of the presence of outlier behaviour in the event data, return complicated and not understandable results on real data. To illustrate the problem we use two examples with synthetic and real event data. In the first example, we generate an event log based on the process model of Fig. 1. Thereafter, different types of noisy behaviour like removing or adding activities are added to these process instances.



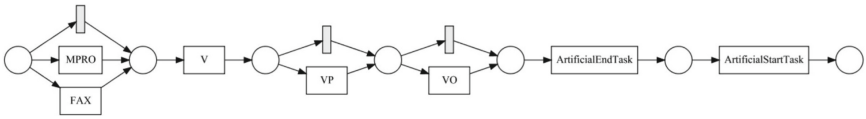
(a) Results of the Inductive Miner [10] on the whole event log.



(b) Result of the Inductive Miner [12] on the event log that filtered beforehand with Matrix Filter [5]. The precision of this model is 0.88 and its fitness is 0.92.



(c) Result of the Inductive Miner [12] on the event log that filtered beforehand with *AFA* [6]. The precision of this model is 0.96 and its fitness is 0.92.



(d) Result of the Inductive Miner [12] on the filtered event log using our proposed filtering method (*SF*).The precision of this model is 1.0 and its fitness is 0.92.

Fig. 3. Discovering process models by the Inductive Miner on the *ICP.anon* event log.

The process model that is discovered by the Inductive Miner [10] and its embedded noise filtering mechanism for this event log, is shown in Fig. 2a. This process model lets all activities happen all in parallel and it means the embedded noisy filtering algorithm in the Inductive Miner is not able to detect and remove all types of added outlier behaviour. We also try to first filter out outlier behaviour and after that apply the Alpha Miner [11] that is a noise sensitive process discovery algorithm on the cleaned event log. Figure 2b shows Anomaly Free Automaton*AFA* filtering method [6] removes many of correct behavoiur during filtering process. Also using *Matrix Filter* [5] that is a probabilistic filtering method in Fig. 2c because it does not detect and remove some outlier behaviour, the result of Alpha miner is not accurate. Finally, Fig. 2d illustrates the result of using the filtering method proposed in this paper. This filter is able to cope with long term dependency, casual - and parallel behaviour, then the discovered process model is completely equivalents to Fig. 1.

As another example, Fig. 3 shows how filtering outlier behaviour in a real event log, i.e. *ICP.anon* [13], reduces the complexity and improves the understandability of a process model. Figure 3a shows a process model that is discovered by the Inductive Miner using its filtering method. Moreover, Fig. 3d indicates the result of applying the same process discovery algorithm (but without its embedded filtering mechanism) on the filtered event log that contains 65% of the process instances of the original event log using our proposed filtering method. Note that, in the process model and also in the event data activity "*ArtificialEndTask*" comes before "*ArtificialStartTask*".

For the real event data, because usually there is not any reference process model, we use quality measures of the discovered process model. *Fitness* and *precision* are measures that are widely used to assess quality of process models [14]. Fitness computes how much behaviour in the event log is also described by the process model. The fitness value equals to 1 indicates that all behaviour in the event data is replayable with the process model. However, precision measures the amount of behaviour described by the process model that is also presented in the event log. The precision value equals to 1 shows that all behaviour in the event log is also presented in the event data. The fitness values of Fig. 3a and d are 1.0 and 0.92 whereas their precision values are 0.64 and 1.0 respectively. This means that the process model in Fig. 3a describes more behaviour that is also presented in the event log, however, in order to do this it is underfitting. Hence, it allows for much more behaviour compared to the model in Fig. 3a. As a consequence, the model in Fig. 3a is more inaccurate. In other words, by sacrificing a little in fitness we reach a more precise model that represent the main stream bahaviour of the event data. Note that, most of the activities in Fig. 3a are rarely present in the event data. Note, Fig. 3c and b also increase the precision by little sacrificing in fitness. However, some casual relations like *VO* eventually happens after *VP* are not detectable with these filtering methods, because they just consider the direct flow relation of activities.

3 Related Work

Early work in process discovery focused solely on the discovery of process models, however, more recently process discovery algorithms have been extended to be able to handle outliers as well [10,15]. However, the extended filtering methods are tailored towards the internal procedures of the corresponding algorithm and hence do not return a filtered event log. This hampers the applicability of these internal filtering techniques as general purpose event log filtering techniques. Some methods like [16,17] are specifically designed to cope with outlier behaviour. However, they do not result in process models with clear execution semantics and again suffer from the previous problem. Many of the commercial process mining tools use simple frequency based filtering methods that is not applicable for some real event data, and moreover, they are again tailored towards their process modeling notation.

Outlier detection for temporal and sequential data is also addressed in the data mining field. For example, [18] surveys on different methods of detecting outliers on discrete sequential data and [19] presents a similar study for temporal data. Also, there are some related techniques that are specifically proposed for the process mining domain. In [20,21], the authors propose filtering techniques that use additional information such as training event data or a reference process model. But, providing a set of training traces that cover all possible outliers or having a reference process model is impractical. In [6], 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 [5], we propose a filtering method that detects outliers based on conditional probabilities of subsequences and their possible following activities. Also, [22] proposes an on-line adjustable probabilistic filtering method that detects outlier behaviour for streaming events. The AFA method is similar to a specific case of probabilistic method that the given subsequence has length 1. [23] presents an entropy-based method to filter chaotic activities i.e., activities that occur spontaneously at any point in the process. Finally, [24] presents a method that detects outliers based on frequent contexts, and subsequently repairs this behaviour instead of removing traces with outlier. In this method sometimes activities that do not happen in reality are added to the event log.

One key difference of process event logs with other general sequential data is the existence of parallel and long term behaviour between activities. All the mentioned previous filtering algorithms just consider the directly follow dependency of activities. However, considering just directly follow relations of activities is not enough to detect outlier behaviour in processes with *parallel*, *casual* and *long term dependency* relations.

For example, the *AFA* method [6] is not able to detect all outliers in such event logs. Moreover, if the subsequence length that is used in conditional probability based methods [5,22] is not enough, they are also not able to detect *parallel* and *long term dependency* relations. For example, if the length of the condition part is 2, $\langle a, f, b, d, f, g, h\rangle$ is accepted as normal behaviour in the event log that corresponds to the process model in Fig. 1. But, it is impossible to execute f two

times or not executing c. Also, if f will execute, activity i have to happen at the end of the process. As another example, $\langle a, d, e, c, g, h\rangle$ also is accepted as a normal behaviour. However, according to Fig. 1, it is not possible to execute c without execution of b beforehand.

4 Preliminaries

This section briefly introduces the basic process mining terminology and notations like event log and multiset, and also discusses concepts such as sequential patterns and sequential rules. 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 $M = [e_1^{k_1}, e_2^{k_2}, \ldots, e_n^{k_n}]$, where for $1 \leq i \leq n$ we have $M(e_i) = k_i$ with $k_i \in \mathbb{N}$. 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 $\mathcal{A}$ denotes the set of all possible activities and let $\mathcal{A}^*$ denote the set of all finite sequences over $\mathcal{A}$. A finite sequence σ of length n over $\mathcal{A}$ is a function $\sigma\colon \{1, 2, \ldots, n\} \to \mathcal{A}$, alternatively written as $\sigma = \langle a_1, a_2, \ldots, a_n\rangle$ where $a_i = \sigma(i)$ for $1 \leq i \leq n$. The empty sequence is written as ϵ. Concatenation of sequences σ and σ' is written as $\sigma \cdot \sigma'$. Moreover, sequence $\sigma' = \langle a'_1, a'_2, \ldots, a'_k\rangle$ is a subsequence of sequence $\sigma = \langle a_1, a_2, \ldots, a_l\rangle$ if we are able to write σ as $\sigma_1 \cdot \langle a'_1, a'_2, \ldots, a'_k\rangle \cdot \sigma_2$, where both σ_1 and σ_2 are allowed to be ϵ. Finally, $\sigma' = \langle a'_1, \ldots, a'_k\rangle$ is an *interval subsequence* of $\sigma = \langle a_1, \ldots, a_l\rangle$, if and only if, we are able to write σ as $\sigma_1 \cdot \langle a'_1\rangle \cdot \sigma_2 \cdot \langle a'_2\rangle \cdot \ldots \cdot \sigma_k \cdot \langle a'_k\rangle \cdot \sigma_{k+1}$. If all $\sigma_2 \ldots \sigma_k$ are equal to ϵ, σ' is also subsequence of σ.

Event logs describe sequences of executed business process activities, typically in context of an instance of the process (referred to as a case), 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. Each of these unique sequences is referred to as a *variant*. In context of this paper, we define event logs as a multiset of sequences of activities, rather than a set of traces describing sequences of unique events.

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

Observe that each $\sigma \in \overline{L}$ describes a *trace-variant* whereas $L(\sigma)$ describes how many traces of the form σ present in the event log. We also define sequential patterns as follows.

如<a, b, c>和<e, e>是<a, b, d, a, e, f, c, e>的两个序列模式

Definition 2 (Sequential pattern). *Let* $\rho = \langle a_1, a_2, \ldots, a_k \rangle \in \mathcal{A}^*$ *and* $\sigma \in \mathcal{A}^*$ *be two non-empty sequences of activities.* ρ *is a sequential pattern in* σ *if and only if* $\exists \sigma'_1, \ldots, \sigma'_{k+1} | \sigma = \sigma'_1.a_1.\sigma'_2. \ \ldots \ .a_k.\sigma'_{k+1}$ *and show it by* $\rho \sqsubseteq \sigma$.

ρ是σ的区间子序列

It is not required that all items in a sequential pattern happen directly after each others. But, it is mandatory that the order of the activities in both sequences is preserved. So, $\rho \sqsubseteq \sigma$ if ρ is an interval subsequence of σ. Also, $\sqsubseteq$ returns a binary value that will be true if the mentioned sequential pattern ρ happens one or more in the sequence σ. For example, $\langle a, b, c \rangle$ and $\langle e, e \rangle$ are two sequential patterns in $\langle a, b, d, a, e, f, c, e \rangle$. The support of a sequential pattern ρ in the event log L is computed as follows.

$$Support(\rho) = \frac{|[\sigma \in L | \rho \sqsubseteq \sigma]|}{|L|} \quad (1)$$

包含σ的迹的个数/日志中迹的个数

When a pattern ρ happens multiple times in a trace, in the general sense, we count it one time. Moreover, we define sequential rules according to the following definition.

σ中从A到C有顺序关系

Definition 3 (Sequential rule). *Let* σ *is a sequence of activities and* $A \subseteq \{a \in \sigma\}$ *and* $C \subseteq \{a \in \sigma\}$ *are two non-empty sets of activities in that sequence. We say that there is a sequential relation from* A *to* C *in* σ *and denote it with* $A \xrightarrow{\sigma} C$, *if and only if* $\exists \sigma_1, \sigma_2 \in \mathcal{A}^* | (\sigma = \sigma_1 \cdot \sigma_2 \wedge A \subseteq \{a \in \sigma_1\} \wedge C \subseteq \{a \in \sigma_2\})$ *where* σ_1 *and* σ_2 *are two subsequences of* σ. A *is the antecedent and* C *is the consequent of the sequential rule. Similarly,* $C \xleftarrow{\sigma} A$, *if and only if* $\{\exists \sigma_1, \sigma_2 | \sigma = \sigma_1.\sigma_2 \wedge A \subseteq \{a \in \sigma_2\} \wedge C \subseteq \{a \in \sigma_1\}\}$.

The order of activities among the antecedent and consequent elements of a rule is not important. However, all activities of the consequent set should at least happen one time after the antecedent activities. For example, in $\sigma = \langle a, b, c, a, b, d, e \rangle$, $\{a, c\} \xrightarrow{\sigma} \{e, b\}$, $\{b\} \xrightarrow{\sigma} \{a, e\}$, and $\{b, c\} \xleftarrow{\sigma} \{a\}$ are possible sequential rules. Also, in both $A \xrightarrow{\sigma} C$ and $C \xleftarrow{\sigma} A$, the A is the antecedent but direction of rules are different. Also, $\xrightarrow{\sigma}$ is a binary function that returns true if the sequential rule occurs at least one time in that sequence. We compute the *Support* and *Confidence* of a sequential rule as follows.

A-->C出现的次数/L中迹的个数

$$Support(A \rightarrow C) = \frac{|[\sigma \in \overline{L} | (A \xrightarrow{\sigma} C)]|}{|L|} \quad (2)$$

在已有A的前提下，C发生的概率即，条件概率

$$Confidence(A \rightarrow C) = \frac{Support(A \xrightarrow{\sigma} C)}{|[\sigma \in \overline{L} | A \subseteq \{a \in \sigma\}]|} \quad (3)$$

=P(C/A)

These measure return a value between 0 and 1. A higher support value means more traces in the event log contain $A \rightarrow C$. The higher confidence value indicates that after occurring the antecedence's activities in a trace, it is more probable that consequence's activities also are present in that trace. Note that, we

just consider the existence of a rule and not the frequency of it in a trace, e.g., in the sequence $\sigma = \langle a, b, c, a, b, d, e \rangle$, $\{a\} \rightarrow \{b\}$ is counted one time even though it happens three times. In the confidence formula, $A \subseteq \sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ if and only if $\{\forall a \in A, \exists k | 1 \leq k \leq n \wedge \sigma_k = a\}$. We simply also define maximum size of antecedent and consequence as $|L_A|$ and $|L_C|$.

5 Filtering Outlier Behaviour Using Sequence Mining

The proposed filtering method requires an input event log and some parameters that are adjustable by the user and consists of three main steps. Firstly, we discover sequential rules and patterns from the event log. Secondly, based on discovered sequential rules and patterns for each trace we search if it contains of any outlier behaviour. Finally, we remove traces with outlier behaviour from the event log. Algorithm 1 shows the pseudo-code of the proposed algorithm.

In the first step, we discover sequential rules and patterns from the given event log. However, we just want to find odd (or low probable) sequential patterns and high probable sequential rules. P is an odd sequential pattern if $Support(P) \leq Sup_O$. The minimum support of sequential patterns or Sup_O is set by the user. So, any sequential pattern with the support value below than Sup_O, will be considered as an odd pattern. For discovering odd patterns, we first discover all possible patterns in the event log (with minimum support equal to 0). Then we remove all sequential patterns with the support value of higher than Sup_O (we call them normal sequential patterns). In an informal way we say $\{OddPatterns\} = \{Patterns_{MinSupport=0}\} - \{Patterns_{MinSupport=Sup_O}\}$.

低概率序列模式，只留**支持度**值低于阈值的序列模式

We also need to discover high probable sequential rules. $A \rightarrow C$ is a high probable sequential rule if $Support(A \rightarrow C) \geq Sup_H$ and $Confidence(A \rightarrow C) \geq Conf_H$. Both Sup_H and $Conf_H$ are also parameters that possibly set by the user and refers to the minimum support and the minimum confidence of high probable rules respectively. We discover high probable sequential rules in both directions (i.e., $A \rightarrow C$ and $C \leftarrow A$). The user also sets the maximum length of sequential patterns (L_P) and size of antecedence (L_A) and consequent (L_C) of sequential rules. These parameters affect the computational complexity of the proposed method. The complexity of discovering sequential patterns (in an event log with m traces and n activities) is $O(m \times n^{L_P})$ and complexity of discovering sequential rules is $O(m \times n^{L_A + L_C})$.

高概率序列规则，支持度和置信度都要高于阈值

After discovering odd sequential patterns and high probable sequential rules, in the second step, we will discover outlier behaviour in process instances based on them. To detect outlier behaviour, for all traces in the event log we apply the following rules:

判定异常行为的规则

- Traces with antecedence of a high probable sequential rule, should also contain the consequence of that rule. 具有高概率序列规则前件的迹，也应包含该规则的后件
- Existence of odd sequential patterns considered as outlier behaviour.

低概率的序列模式的存在被视为异常行为。

Algorithm 1. Filtering Event Log using Sequential Mining

```
procedure DISCOVERING SEQUENTIAL RULES AND PATTERNS (L, S_O, S_H, C_H, L_P, L_A, L_C)
    ProbableRules ← {All sequential rules according to S_H, C_H, L_A, L_C }
    AllPatterns ← {All sequential patterns with length L_P }
    NormalPatterns ← {All sequential patterns with length L_P and minimum support of S_O }
    OddPatterns = AllPatterns − NormalPatterns
     return (ProbableRules, OddPatterns )

procedure DISCOVERING AND REMOVING OUTLIER(L, ProbableRules, OddPatterns)
     FilteredLog ← {}
    OutlierFlag = 0
    for each Trace T ∈ L do
        for each Rule R ∈ ProbableRules do
            if (T contains Antecedence of R & does not hold R ) then
                OutlierFlag= 1
        for each Pattern P ∈ OddPatterns do
            if (T contains P ) then
                OutlierFlag= 1
        if (OutlierFlag ! = 1) then
            FilteredLog ← T
    return (FilteredLog)
```

Note that if $A \xrightarrow{\sigma} C$ is false it is not necessary results in that there is outlier behaviour in the trace σ. But, if a trace contains an antecedence part of a high probable sequential rule (A), it should hold that sequential rule completely ($A \xrightarrow{\sigma} C$ equals true), otherwise, according to our definition, this trace contains outlier behaviour.

In general, odd sequential patterns are used to detect inserted or wrong ordered activities in traces. For example, in Fig. 1, for the trace $\langle a, e, b, d, c, f, g, h\rangle$, we have the odd pattern $\langle e, f\rangle$. So, the existence of both the e and f is not normal in this trace and it is detectable only when we consider long term follow relations. Moreover, high probable sequential rules are used to detect possible removed activities in traces. For example in Fig. 1, for the trace $\sigma = \langle a, b, d, f, g, i\rangle$ and having $\{a, b\} \xrightarrow{\sigma} \{c\}$, we find that the activity c does not occur after b in this trace. Therefore, we detect that this trace contains outlier behaviour.

Finally, in the last step, all traces that contain outlier behaviour will be separated from the event log. Therefore, in the filtered event log that will be returned back to the user, there is not any trace that contains outlier behaviour. There is also an option that we return only traces that contain outlier behaviour. This option is useful when we want to analyze infrequent and abnormal behaviour in the process instead of having the general view and the mainstream of the process.

Observe that, according to the definitions of sequential patterns and sequential rules, it is not required that activities in the rules and patterns are executed directly after each other. For example, an odd pattern $\langle e, i\rangle$ indicates that if activity e happens, activity i should not happen anywhere after e in that trace, otherwise we consider it as outlier behaviour. Such long distance (or indirect) follow relations that are not considered in previous filtering methods, are helpful to detect outlier behaviour in event logs with the heavy presence of *parallel* and *casual* relations.

Implementation. To be able to apply the proposed filtering method on event logs and to find its helpfulness, we implemented the *Sequential Filter* plug-in (*SF*) in the ProM framework[1] [7]. This plug-in takes an event log as an input with some parameters and outputs a filtered event log. In this implementation, for discovering high probable sequential rules we use the *rule growth* algorithm [4]. Also for discovering sequential patterns we use *prefixspan* [3] Both these sequential mining algorithms have been implemented in the SPMF tool [25].

To apply our proposed method on various event logs with different thresholds and applying different process mining algorithms with various parameters, we ported the *Sequential Filter* (*SF*) plug-in to RapidProM. RapidProM is an extension in the RapidMiner tool that combines scientific workflows with a range of (ProM-based) process mining algorithms [26].

6 Evaluation

To evaluate our proposed filtering method we applied it on real and synthetic event logs. The main goal of this evaluation is assessing the capability of the sequential filter (*SF*) compared to other existing general purpose filtering methods from two aspects:

- Improvement rate of discovered process models.
- Level of detectability of filtering methods.

For the first aspect, we consider the improvement of the discovered process model using both real and synthetic event data. However, for the second aspect, we just consider the synthetic event logs, because we need reference models as the ground truths that indicate which of the traces contain outlier and which ones are clean. Note that, the filtered event log is just used for discovering purpose and for measuring the quality of the discovered model, the original event log has been used. In Table 1, we present the event logs that are used in this evaluation with some characteristics of them. All of these real event logs are accessible via https://data.4tu.nl/repository/collection:event_logs_real and http://www.processmining.be/actitrac. The number of activities in these event logs are different from 6 to 70. For some event logs like $BPIC\_2018\_Inspection$ there are few but frequent variants, however, in the $Hospital\_Billing$ event log each most of variants are unique or happen just one time. We select these event logs to check the feasibility of filtering methods and how they are able to improve process discovery algorithms results for event logs from different domains. For the synthetic event, we first designed the reference process models. As we claimed that previous filtering methods are not able to detect all outlier behaviour in event logs with a heavy presence of parallel and/or long term dependency behaviour, the process models are designed with this behaviour. The reference process models are presented in Fig. 5. After generating corresponding event logs from these reference process models, we injected different percentages

[1] Sequential filter plugin svn.win.tue.nl/repos/prom/Packages/LogFiltering.

of artificial outlier behaviour to the original event logs. In this regard, we insert, remove and change the order of activities. The reference models are used to determine which traces are clean and which one contain outlier behaviour.

Table 1. Some details about the event logs that are used in the experiment.

Event log	Real/Synthetic	Activity #	Event #	Trace #	Variant #
BPIC_2012_Application	Real	10	60,849	13,087	17
BPIC_2012_Offer	Real	7	31,244	5,015	168
BPIC_2012_Work	Real	6	72,413	9,658	2263
BPIC_2017_Offer	Real	8	193,849	42,995	16
BPIC_2018_Department	Real	6	46,669	29,297	349
BPIC_2018_Financial	Real	36	262,200	13,087	4,366
BPIC_2018_Inspection	Real	26	197,717	5,485	3,190
BPIC_2018_Parcel	Real	10	132,963	14,750	3,613
Hospital_Billing	Real	18	451,359	100,000	1,020
Road_Fines	Real	11	561,470	150,370	231
Sepsis	Real	16	15,214	1,050	846
TSL.anon	Real	40	83,286	17,812	2,551
MCRM.anon	Real	22	11,218	956	212
ICP.anon	Real	70	65,653	12,391	1,411
KIM.anon	Real	18	124,217	24,770	1,174
High_Variants	Synthetic	20	75,915	5,000	4,668
High_Parallel	Synthetic	19	95,000	5,000	5,000
Long_Term_Dependency	Synthetic	14	50,000	5,000	3

To evaluate the quality of discovered process models, we use *fitness* and *precision*. Fitness computes how much behaviour in the event log is also described by the process model. Precision measures how much behaviour that is described by the discovered 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 [27]. Sometimes, putting aside a small amount of behaviour causes a slight decrease in fitness, whereas precision increases much more. Therefore, to evaluate improvement of discovered process models, we use the *F-Measures* metric that combines fitness and precision: $\frac{2 \times Precision \times Fitness}{Precision + Fitness}$. Moreover, to assess outlier detectability of filtering methods, we map the problem to a classification problem. In other words, we should detect whether a trace contains outlier behaviour or whether it is clean. Therefore, we use the well-known classification measure *F1-score* [28] that combines *Precision* and *Recall* of filtering methods' outlier detectability on noisy event logs. So to have a high F1-score the filtering method should detect outlier as much as possible (i.e., Recall) and at the same time, most of the removed traces indeed contain outlier behaviour (i.e., Precision).

Also because the proficiency of all current filtering methods depends on their adjusted parameters, for each of them a grid search method is used to find their best result. Therefore, just the best result for each filtering method is depicted. In this regard, we use *AFA* method with 50 different thresholds from 0 to 1 and 25 different thresholds for *MF* with subsequence length equals 2 to 4. For process discovery purpose, we applied the basic Inductive Miner [12] on filtered event logs, because it always returns sound process models that makes it feasible to compute *fitness* and *precision* for discovered process models. Also, we used the Inductive Miner [10] with its embedded filtering mechanism with 50 different thresholds from 0 to 1.

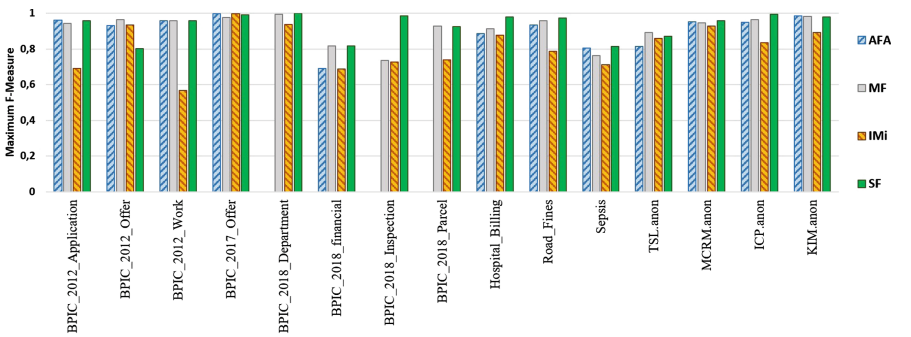
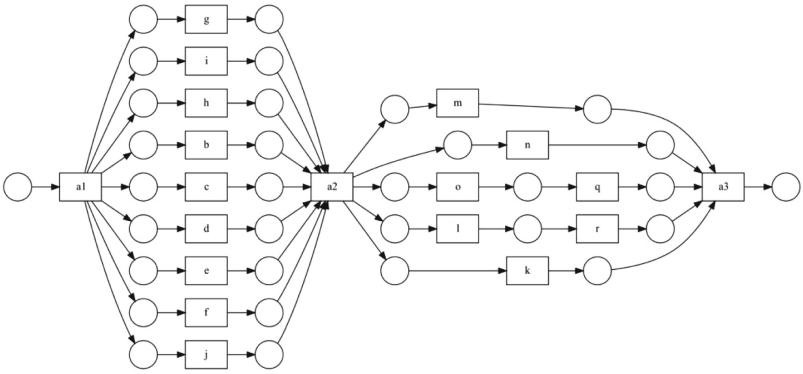
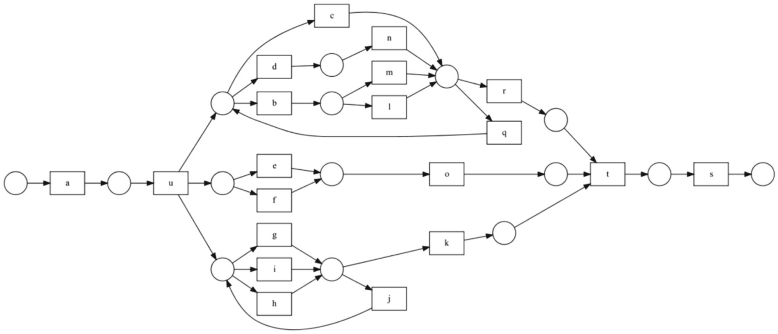


Fig. 4. The best F-Measure for using different filtering methods on some real event logs.

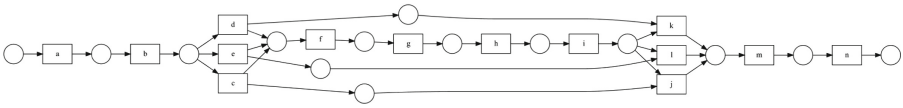
Figure 4, indicates the best F-Measures of discovered process models when different filtering methods are used to remove outlier behaviour. *AFA* and *MF* are consequently related to *Anomaly Free Automaton* [6] and *Matrix Filter* [5] methods and *SF* refers to the sequence mining based filtering method. We also compare these results with *IMi* that refers to using the Inductive Miner with its embedded filtering mechanism. Note that for the other filtering methods we just applied the basic *Inductive Miner* without its filtering mechanism. For most of the event logs, *SF* returns the best F-Measure. However, for the event log $BPIC\_2012\_Offer$ compare to other methods it does not perform well. It is because in this event log there is a long loop, i.e., that is a loop in the process model with more than 2 activities. In general, an existence of long loops in the given event log has negative effects on the outlier detectability of *SF*. Also, in the heavy presence of choice (Xor) behaviour it will be more probable that our proposed method is not able to detect all removed/missed activities. We will discuss limitations of the proposed method in Sect. 6. Observe, for some event logs like *Sepsis* even with filtering outliers we could not reach a process model with *F-Measure* value near 1. It is caused by characteristics of the event log. We found that if a rate of $1 - \frac{Variant\#}{Trace\#}$ be lower, then it would be harder to discover a process model with high precision and fitness at the same time.



(a) A process model with heavy presence of parallel behaviour (*High_Parallel*)



(b) A process model with all different types of behaviour and possibility of having high number of variants (*High_Variant*)



(c) A process model with long term dependency behaviour ($Long\_Term\_Dependency$)

Fig. 5. Synthetic process models with different types of behaviour.

Also, the *AFA* filtering method is not able to filter $BPIC\_2018\_Parcel$ and $BPIC\_2018\_Inspection$. So, we have not corresponding results for them on Fig. 4.

We repeat this experiment for the synthetic logs. The best *F-Measure* discovered process models using different filtering methods on these synthesis event logs are given in Fig. 6. For *High_Parallel* event log with increasing the percentage of injected outlier behaviour just by using *SF* we are able to reach an acceptable process model. This is because except *SF*, other filtering methods use direct follow dependencies and it is not enough to detect all outliers when we have a heavy presence of parallel behaviour in an event log.

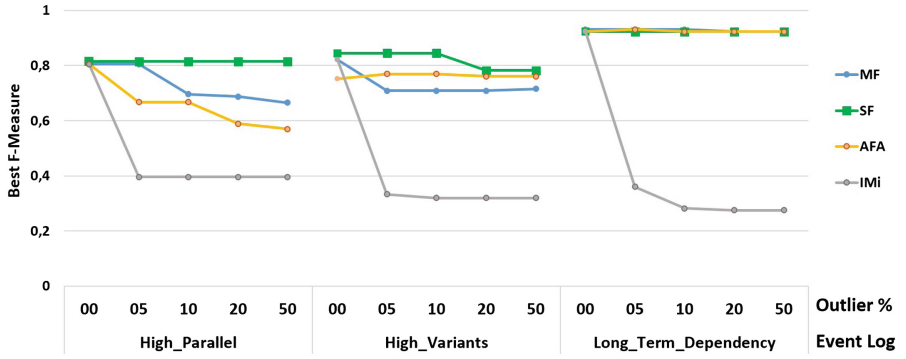


Fig. 6. The F-Measure of applying different filtering methods on synthetic event logs.

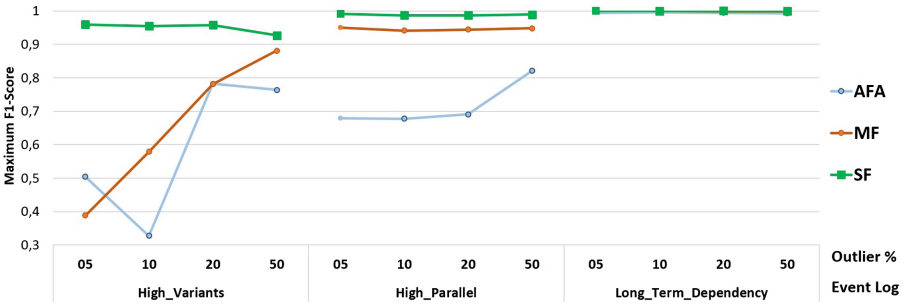


Fig. 7. The F1-Score of applying different filtering methods on synthetic event logs.

As shown in this experiment, even for the original event logs (without added outlier behaviour) there is no process model with perfect *F-Measure*. This problem is caused by process model discovery algorithm -here the *Inductive Miner*- that is not able to show long-term dependencies and also the definition of precision that does not return the value 1 even for the original process model and the original event log. As we use escaped edge definition [29] for computing precision, it is interesting to note that for complicated process models (with heavy presence of *parallel*, *loop* and *casual* relations), even the precision of the perfect process model will be lower than 1.

In the next experiment, using synthetic event logs we want to assess how different filtering methods are able to detect traces with injected outlier behaviour. The best *F1-Score* for detectability of different filtering methods are indicated in Fig. 7. For *Long_Term_Dependency* event log, the *F1-Score* for *SF* is 1 but for *AFA* and *MF* it equals to 0.993 and 0.998 respectively. As in Fig. 6, we used the Inductive Miner algorithm and it is not able to show long-term dependencies, there is no difference between the output of these filtering methods. However, if we use other algorithms like Alpha++ [30], we will discover a model exactly like Fig. 5c that its *F-Measure* value will be equal to 1.

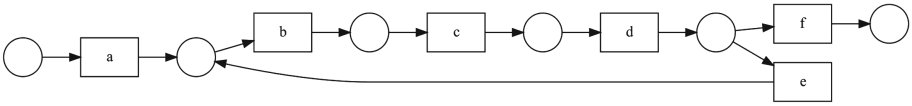


Fig. 8. A process model with loop behaviour.

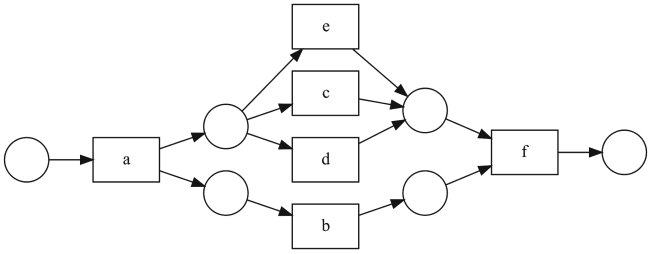


Fig. 9. A process model with *Xor* behaviour.

The experiment results show that all filtering methods improve the process discovery results. Also, results indicate that using sequential patterns and rules, we are able to detect outlier behaviour in process models with a heavy presence of parallel, long term-dependencies, and casual behaviour.

Limitations. Our proposed filtering method uses sequential patterns and rules that cause to consider indirectly (long term) follow relations. However, it has limitations because it does not benefit from directly follow relations. The first limitation is that when we have loop behaviour, the proposed method is not able to detect some outlier behaviour relates to activities that are participated in the *do* part of the loop. For example, in Fig. 8, activities b, c and d construct the *do* part and e is the *redo* part of the loop. In this process model, it is possible that the filtering method does not detect outlier behaviour in $\langle a, b, c, d, e, d, b, c, c, f \rangle$. Due to the existence of loop behaviour, it is more likely that we consider $\langle c, c \rangle$, $\langle c, d \rangle$, and $\langle d, b \rangle$ as normal sequential patterns. Therefore, it would be challenging to detect outlier behaviour that loop activities.

Another limitation of the proposed filtering method is that the presence of *Xor* behaviour has negative effects on the proficiency of it. For example, in Fig. 9 after the execution of activity a, one of activities c, d, and e execute (in parallel with activity b). The sequence filtering method is not able to detect *removed* activities that are participated in the *Xor* behaviour. But, added and misordered activities are detectable even with the heavy presence of *Xor* behaviour. Note that, by decreasing the minimum support of high probable sequential rules such outlier behaviour is detectable, but it causes to detect some of normal behaviour as outlier behaviour too. For example, in Fig. 9 to detect outlier behaviour of $\langle a, b, e \rangle$, we need to consider a sequential rule like $\{a\} \leftarrow \{c, f\}$ as a high probable rule that will cause to remove all traces that do not contain activity c.

Filtering methods like [5] and [6] that use direct follow relations are easily able to detect outlier behaviour in event logs with *loop* and *Xor* relations. Therefore, it seems that if there is a filtering method that uses both of direct and undirected follow relations it is able to detect outlier behaviour more accurately.

7 Conclusion

Process mining allows us gain insights into the actual execution of business processes using available event data. Most of process mining algorithms work under the assumption that the input data is free of outlier behaviour. However, real event logs typically contain outlier (i.e., noise and infrequent) behaviour which leads to imprecise/unusable process mining results. Detecting and removing such behaviour in event logs 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 filtered event log. It exploits sequence mining algorithms to discover sequential patterns and rules. Using such information we able to discover flow relation of activities over long distances. By applying sequence mining methods, it does not just rely on the direct flow of activities and we are able to detect outlier behaviour in event logs with a heavy presence of *parallel*, *casual* or *long term dependency* behaviour.

To evaluate the proposed filtering method, we developed a plug-in in the ProM platform and the RapidProM. As presented, we have applied this method to several real event logs and compared it with other state-of-the-art process mining specific data filtering methods. Additionally, we applied the proposed method on synthetic event logs. The results indicate that the proposed filtering approach is able to detect 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 the sequence filter method outperforms other state-of-the-art process mining filtering techniques as well as the Inductive Miner algorithm with its embedded filtering mechanism for some real event logs.

As future work, we aim to conduct larger experiments to find out in which situations a particular filtering mechanism works better. We also aim to estimate the filtering parameters based on features of the given event log.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Maruster, L., Weijters, A.J.M.M., van der Aalst, W.M.P., van den Bosch, A.: A rule-based approach for process discovery: dealing with noise and imbalance in process logs. Data Min. Knowl. Discov. **13**(1), 67–87 (2006)
3. Han, J., et al.: PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceedings of the 17th International Conference on Data Engineering, pp. 215–224 (2001)

4. Fournier-Viger, P., Wu, C.W., Tseng, V.S., Cao, L., Nkambou, R.: Mining partially-ordered sequential rules common to multiple sequences. IEEE Trans. Knowl. Data Eng. **27**(8), 2203–2216 (2015)
5. Sani, M.F., van Zelst, S.J., van der Aalst, W.M.P.: Improving process discovery results by filtering outliers using conditional behavioural probabilities. In: Teniente, E., Weidlich, M. (eds.) BPM 2017. LNBIP, vol. 308, pp. 216–229. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74030-0_16
6. 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), 300–314 (2017)
7. 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), 2 (2009)
8. 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)
9. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall Inc., Englewood Cliffs (1981)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013. LNBIP, vol. 171, pp. 66–78. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06257-0_6
11. 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), 1128–1142 (2004)
12. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
13. De Weerdt, J., vanden Broucke, S., Vanthienen, J., Baesens, B.: Active trace clustering for improved process discovery. IEEE Trans. Knowl. Data Eng. **25**(12), 2708–2720 (2013)
14. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: Meersman, R., et al. (eds.) OTM 2012. LNCS, vol. 7565, pp. 305–322. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33606-5_19
15. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Avoiding over-fitting in ILP-based process discovery. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 163–171. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_10
16. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: CIDM (2011)
17. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_24
18. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection for discrete sequences: a survey. IEEE Trans. Knowl. Data Eng. **24**(5), 823–839 (2012)
19. Gupta, M., Gao, J., Aggarwal, C.C., Han, J.: Outlier detection for temporal data: a survey. IEEE Trans. Knowl. Data Eng. **26**(9), 2250–2267 (2014)
20. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning structured event logs: a graph repair approach. ICDE **2015**, 30–41 (2015)

21. Cheng, H.J., Kumar, A.: Process mining on noisy logs—can log sanitization help to improve performance? Decis. Support Syst. **79**, 138–149 (2015)
22. van Zelst, S.J., Fani Sani, M., Ostovar, A., Conforti, R., La Rosa, M.: Filtering spurious events from event streams of business processes. In: Krogstie, J., Reijers, H.A. (eds.) CAiSE 2018. LNCS, vol. 10816, pp. 35–52. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91563-0_3
23. Tax, N., Sidorova, N., van der Aalst, W.M.P.: Discovering more precise process models from event logs by filtering out chaotic activities. J. Intell. Inf. Syst. 1–33 (2018)
24. Fani Sani, M., van Zelst, S.J., van der Aalst, W.M.P.: Repairing outlier behaviour in event logs. In: Abramowicz, W., Paschke, A. (eds.) BIS 2018. LNBIP, vol. 320, pp. 115–131. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93931-5_9
25. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.: SPMF: a Java open-source pattern mining library. J. Mach. Learn. Res. **15**(1), 3389–3393 (2014)
26. Bolt, A., de Leoni, M., van der Aalst, W.M.P.: Scientific workflows for process mining: building blocks, scenarios, and implementation. STTT **18**(6), 607–628 (2016)
27. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A robust F-measure for evaluating discovered process models. In: Proceedings of the CIDM, pp. 148–155 (2011)
28. Makhoul, J., Kubala, F., Schwartz, R., Weischedel, R., et al.: Performance measures for information extraction. In: Proceedings of DARPA Broadcast News Workshop, Herndon, VA, pp. 249–252 (1999)
29. Munoz-Gama, J., Carmona, J.: Enhancing precision in process conformance: stability, confidence and severity. In: 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 184–191. IEEE (2011)
30. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. **15**(2), 145–180 (2007)