Applying Sequence Mining for Outlier Detection in Process Mining

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

[1]Process and Data Science Chair, RWTH Aachen University
Ahornstraße 55, Erweiterungsbau E2, 52056 Aachen, Germany
[2]Fraunhofer FIT, Birlinghoven Castle, Sankt Augustin, Germany
{fanisani,wvdaalst,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. We show that, 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.

Key words: Process Mining · Sequence Mining · Event Log Filtering · Event Log Preprocessing · Sequential Rule Mining · Outlier Detection

1 Introduction

Process mining bridges the gap between traditional data analysis techniques like data mining and business process management analysis [1]. 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 [2]. In each case, i.e. process discovery, conformance checking and process enhancement, 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 [3]. 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 may be supposed to happen, 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 of 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 is costly and time-consuming and also needs business/domain knowledge of the data.

In this paper, we focus on improving process discovery results by applying an automated event data filtering, i.e., filtering the event log prior to apply any process discovery algorithm, without significant human interaction.

Sequence mining algorithms like [4, 5] help us to discover and consider existing sequential rules and patterns in event data. Advantaging from sequential rules and patterns, long distance and indirect flow relation will be considered. As a consequence, the proposed filtering method is able to detect outlier behaviour even in event data with lots of concurrencies, and long-term dependency behaviour. The presence of this type of patterns is shown to be hampering the applicability of automated general purpose filtering techniques like [6, 7]. In our proposed filtering method, any process instant in the inputted event data that contains an outlier behaviour is removed from the outputted (filtered) event data.

By using the `ProM` (`http://www.promtools.org`) [8] based extension of `RapidMiner` (`http://www.rapidminer.com`), i.e. `RapidProM` [9], we study the effectiveness of our approach, using synthetic and real event data. The results of our experiments show that our approach adequately identifies and removes outlier behaviour, and, as a consequence increases the overall quality of process discovery results. Additionally, we show that our proposed filtering method detects outlier behaviour better compared to existing event log filtering techniques for event data with heavy parallel and long-term dependency.

The remainder of this paper is structured as follows. Section 2 motivates the need for applying filtering methods on event logs. In Section 3, we discuss related work, then after explaining some preliminaries in Section 4, in Section 5, we describe our proposed method. Details of the evaluation and corresponding results are given in Section 6. Finally, Section 7 concludes the paper and presents some 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 [2], e.g., in Figure 1 we use a Petri net [10] to describe a process. This process model indicates that activity a starts the process and either h or i ends it. Also, it describes activities b, c, d, e, and f are in a parallel relation. This means that we are able to execute these activities in any order in-between activity a and g. However, between b and c there is a casual relation, that means activity c is only allowed to occur

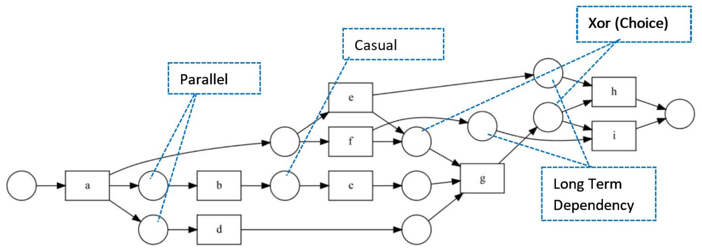
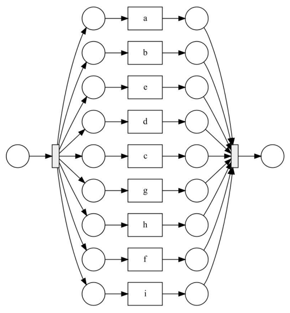


Fig. 1: An example of a process model with *Xor*, *parallel*, *long distance dependency* and *casual* behaviour.

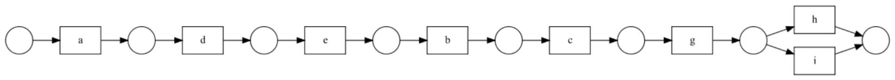
after the execution of activity *b*, but in between them other activities like *d* and *e* are allowed to be executed. Between *e* and *f* and also among *h* and *i* there are *Xor* (or choice) relations. The *Xor* relation indicates that just one of the activities is allowed to be executed. For example, after occurrence the of activity *g* either *e* or *f* is allowed to end the process. Finally, the process model indicates there are long term dependency relations between activities *e* and *h* and also between *f* and *i*. It means that, if in the middle of the process *e* happens, only activity *h* end the process.

Process discovery algorithms aim to discover process models, i.e. models such as Figure 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 return complicated and not understandable results on real data. Most of the times this is because of the presence of outlier behaviour in the event data, otherwise, in case the event data is “clean”, often these process discovery algorithms do result in simple and accurate models. For example, we generate an event log with 5000 process instances based on the process model of Figure 1. This event log like Figure 1 just let 24 unique bahaviour (from start to end part). Thereafter, we randomly add different types of noisy behaviour like removing or adding activities in these process instances. The process model that is discovered by the Inductive Miner [11] and its embedded noise filtering mechanism, is shown in Figure 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 noisy behaviour. We also try to firstly filtering out outlier behaviour and after that applying the Alpha Miner [**?**] as a noise sensitive process discovery algorithm on the cleaned event log. Figure 2b shows the discovered model using this approach when *AFA* filtering method is applied on noisy event log. It means that many of correct behavoiur also removed during filtering process. If we use the *Matrix Filter* for removing outlier behaviour, the discovered model is Figure 2c. It is not a sound model and also because it does not detect and remove some noisy behaviour, the result of a noise sensitive process model is not accurate. Moreover, Figure 2d illustrates the result of using our proposed filtering method that detects long term - and parallel behaviour, and then applying the Alpha algorithm that is completely equivalents to Figure 1.

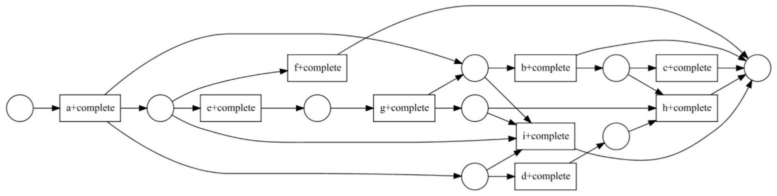
As another example, in Figure 3 we show how filtering outlier behaviour in an event log reduces the complexity and improves the understandability of a process model of a



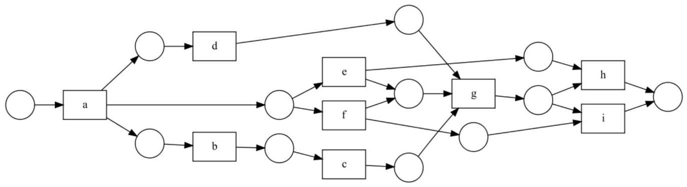
(a) Results of the Inductive Miner [11] on the raw event data.



(b) Result of the Alpha Miner [12] on the event log that filtered beforehand with *AFA* [7].



(c) Result of the Alpha Miner on the filtered event log using matrix filter [6] 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 Figure 1.

real event data, i.e. *ICP.anon* event log [13]. Figure 3a shows a process model that is discovered by the Inductive Miner using its embedded filtering method [11]. For this event log, whereas Figure 3d shows 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 activity "*ArtificialEndTask*" comes before "*ArtificialStartTask*". Although, it seems to be different according to the meaning of the activities, in the event data they presented in this way.

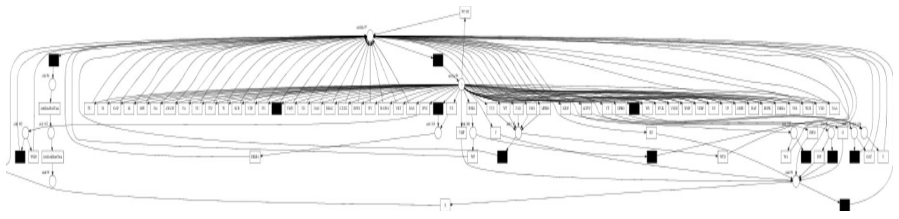
There are two quality measures that are defined for measuring the behavioural quality of process models, i.e. *fitness* and *precision* [14]. Fitness computes how much behaviour in the event log is also described by the process model. However, precision measures the amount of behaviour described by the process model that is also presented in the event log. The fitness values of Figure 3a and Figure 3d are 1.0 and 0.92 whereas their precision values are 0.64 and 1.0 respectively. This means that the process model in Figure 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 Figure 3d. As a consequence, the model in Figure 3a is more inaccurate and ambiguous. In other words, by sacrificing a little in fitness we could reach a more precise model that represent the main stream bahaviour of the event data. Note, most of the activities in Figure 3a are rarely presented in the event data. Even Figure 3a shows all possible behaviour in the event data, but it lets lots of other unseen behaviour too. Note, Figure 3c and Figure 3b that are discovered by other filtering methods also increase the precision by little sacrificing in fitness. However, some casual relations like *VO* eventually happens after *VP* are not detectable with filtering methods like *MF* [6] and *AFA* [7], because they just consider the direct flow relation of activities. But, to consider some kind of relations like f and i or b and d we need to also consider long distance (or indirect) flow relations.

3 Related Work

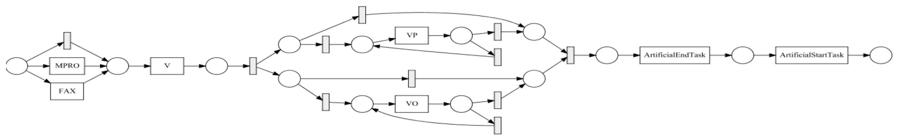
Early work in process discovery focused solely on the discovery of process models, however, more recently process discovery algorithms have been extended to be able to handle outliers as well [11, 16]. However, these extended filtering methods are tailored towards the internal procedures of the corresponding algorithm and hence do not return a cleaned/filtered event log. This hampers the applicability of these internal filtering techniques as general purpose event log filtering techniques. Some other process discovery algorithms like [17, 18] are specifically designed to cope with noisy and infrequent behaviour. However, these algorithms do not result in process models with clear execution semantics and again suffer from the previous problem.

Most of the commercial process mining tools use simple frequency based filtering methods, and moreover, are again tailored towards their proprietary process modeling notation. There are also some basic filtering plug-ins developed in ProM [8] based on activity frequencies and users inputs. For example, the user could filter event data based on process instances' - or events' attribute values.

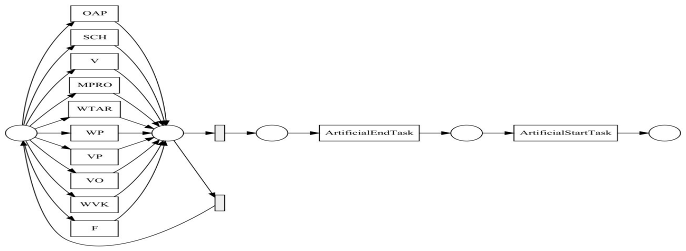
Outlier detection for temporal and sequential data is also addressed in the data mining field. For example, [19] surveys on different methods of detecting outliers on discrete sequential data and [20] presents a similar study for temporal data. Also, there are some related techniques that are specifically proposed for process mining domain. In [21,22] the authors propose filtering techniques that use additional information such as training event data or a reference process model. In reality, providing a set of training traces that cover all possible outliers or having a reference process model is impractical. In [7], 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 [6], we propose a filtering method that detects outliers based on conditional probabilities of subsequences and their possible following activities. Also, in [23] an adjustable on-line filtering method is proposed that detects outlier behaviour for streaming event data. [24] presents an entropy-based method to filter chaotic activities i.e., activities that occur spontaneously at any point in the process. Finally, in [25] a repair method is presented that detects outliers based on frequent contexts in traces, and



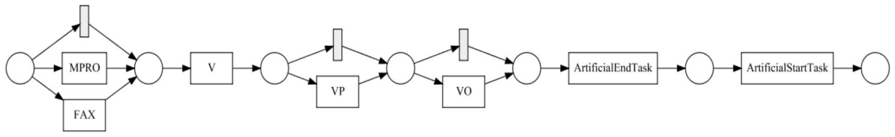
(a) Results of the Inductive Miner [11] on the whole event log.



(b) Result of the Inductive Miner [15] on the event log that filtered beforehand with Matrix Filter [6]. The precision of this model is 0.88 and its fitness is 0.92.



(c) Result of the Inductive Miner [15] on the event log that filtered beforehand with *AFA* [7]. The precision of this model is 0.96 and its fitness is 0.92.



(d) Result of the Inductive Miner [15] 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: Process models discovered by the Inductive Miner on the *ICP.anon* event log [13].

subsequently repairs this behaviour instead of removing traces with outlier. The drawback of the last approach is sometimes activities that do not happen in reality added to the event log.

One of the 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 filtering algorithms just considering 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, *AFA* method [7] is not able to detect outliers in such event logs. Moreover, if the subsequence length that is used in conditional probability based methods [6, 23] will not be enough, they also not able to detect *parallel*, *casual* and *long term dependency* relations. For example, if the length of the condition part be 2, $\langle a, f, b, d, f, g, h\rangle$

will be accepted as normal behaviour. 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 will be accepted as a normal behaviour. However, according to Figure 1, it is not possible to execute *c* without execution of *b* beforehand.

In this paper, we propose a filtering method that uses sequential relations to detect outlier behaviour. For example, in Figure 1, we are able to detect and apply rules like h should happens eventually after *e*. Also, using this filtering method we are able to consider sequential rules like happening activity*c* anytime before activity *b* as an outlier behaviour that is not detectable with other filtering methods.

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.

4.1 Basic Notations and Definitions

Given a set X, a multiset M over X is a function $M\colon X \rightarrow \mathbb{N}_{\geq 0}$, 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 $k_i \in \mathbb{N}$. 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 $\sigma\colon \{1, 2, ..., n\} \rightarrow \mathcal{A}$, alternatively written as $\sigma = \langle a_1, a_2, ..., a_n\rangle$ where $a_i = \sigma(i)$ for $1 \leq i \leq n$. The empty sequence is also written as ϵ. Concatenation of sequences σ and σ' is written as $\sigma \cdot \sigma'$. We let $hd\colon \mathcal{A}^* \times \mathbb{N}_{\geq 0} \nrightarrow \mathcal{A}^*$ with, given some $\sigma \in \mathcal{A}^*$ and $k \leq |\sigma|$, $hd(\sigma, k) = \langle a_1, a_2, .., a_k\rangle$, i.e., the sequence of the first k elements of σ. Note that $hd(\sigma, 0) = \epsilon$. Symmetrically, $tl\colon \mathcal{A}^* \times \mathbb{N}_{\geq 0} \nrightarrow \mathcal{A}^*$ is defined as $tl(\sigma, k) = \langle a_{n-k+1}, a_{n-k+2}, ..., a_n\rangle$, i.e., the sequence of the last k elements of σ. Again, $tl(\sigma, 0) = \epsilon$. Moreover, sequence $\sigma' = \langle a'_1, a'_2, ..., a'_k\rangle$ is a subsequence of sequence $\sigma = \langle a_1, a_2, ..., a_l\rangle$ if we are able to write σ as $\sigma_1 \cdot \langle a'_1, a'_2, ..., a'_k\rangle \cdot \sigma_2$, where both σ_1 and σ_2 are allowed to be ϵ. Finally, $\sigma' = \langle a'_1, a'_2, ..., a'_k\rangle$ is an intervally subsequence (*IS*) of sequence $\sigma = \langle a_1, a_2, ..., a_l\rangle$, if and only if, we are able to write σ as $\sigma_1 \cdot \langle a'_1\rangle \cdot \sigma_2 \cdot \langle a'_2\rangle \cdot ... \cdot \sigma_k \cdot \langle a'_k\rangle \cdot \sigma_{k+1}$. If all $\sigma_2..\sigma_k$ 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

Table 1: 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)	*Aria*	*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)	*Aria*	*2017-04-08:11.13*
1	*Send to manager*(e)	*Dina*	*2017-04-08:13.09*
1	*accept request* (f)	*Fatima*	*2017-04-08:16.05*
1	*mail decision*(h)	*Haan*	*2017-04-08:16.18*
...	...	...	...

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 a *variant*. An example event log, adopted from [2], is presented in Table 1.In this table consider all activities related to *Case-id* value 1. Sara *registers a request*, after which Aria *examines it thoroughly*. William *checks the ticket and resources* after which Dina performs the *send to manager* activity and Fatima *accept the request*. Finally, Haan *mail decision* to the requester. The execution of an *activity* considering in a business process is referred to as an *event*. A sequence of events, e.g., the sequence of events related to case *1*, is referred to as a *trace*. The example process instance is presented as $\langle a, b, c, d, e, f, h\rangle$.

In the 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 $\sigma \in L$ is a variant.*

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

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

Note, it is not required that all items in a sequential pattern happens directly after each others. However, it is mandatory that the order of the activities in both sequences is preserved. In other words, we say that $\rho \sqsubseteq \sigma$ if ρ be an intervally 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 $\langle a, b, d, a, e, f, c, e\rangle$.

We define the support of a sequential pattern ρ in the event log L by the following formula.

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

In other word we count in how many traces of the event log, pattern ρ is a sequential pattern. Note, when a pattern ρ happens multiple times in a trace, in the general sense, we count it 1 time. Moreover, we define sequential rules according to the following definition.

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 $\{\exists \sigma_1, \sigma_2 | \sigma = \sigma_1.\sigma_2 \wedge A \subseteq \{a \in \sigma_1\} \wedge C \subseteq \{a \in \sigma_2\}\}$ 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, we are able to define $C \xleftarrow{\sigma} A$, if and only if $\{\exists \sigma_1, \sigma_2 | \sigma = \sigma_1.\sigma_2 \wedge A \subseteq \{a \in \sigma_2\} \wedge C \subseteq \{a \in \sigma_1\}\}$*

Note, the order of activities among the antecedent and consequent 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 $\sigma = \langle a, b, c, a, b, d, e \rangle$, $\{a, c\} \xrightarrow{\sigma} \{b, e\}$ and $\{b\} \xrightarrow{\sigma} \{a, e\}$ are two possible sequential rules. For this sequence σ, $\{a\} \xleftarrow{\sigma} \{b\}$ and $\{b, c\} \xleftarrow{\sigma} \{a\}$ are also two possible sequential rules. Note that, in both $A \xrightarrow{\sigma} C$ and $C \xleftarrow{\sigma} A$, the set A is the antecedent but direction of rules are different. Also, $\xrightarrow{\sigma}$ is a binary function that returns true if a given sequential rule occurs at least one time in that sequence. We compute the *Support* of a sequential rule as follows.

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

This measure returns a value between 0 and 1. A higher value means more traces in the event log hold $A \rightarrow C$. Note that, we just consider the existence of a rule not the frequency of it in a trace. For example, in the sequence $\sigma = \langle a, b, c, a, b, d, e \rangle$, $\{a\} \rightarrow \{b\}$ counted one time even it happens 3 times. In the same way, a confidence of a sequential rule is defined as the number of traces that hold that rule divided by the number of traces that have the antecedent's activities.

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

Note, $A \subseteq \sigma = \langle \sigma_1, ..., \sigma_n \rangle$ if and only if $\{\forall a \in A, \exists k | 1 \leq k \leq n \wedge \sigma_k = a\}$. The confidence returns a value between 0 and 1 and a higher value indicates that after happening the antecedence's activities in a trace, it is more probable that consequence's activities also will be happened in that trace. We simply also define maximum size of antecedent and consequence as $|L_A|$ and $|L_C|$. Based on these definitions, in the next section, we show how by using sequential rules we detect outlier behaviour in the given event log.

5 Filtering Outlier behaviour using Sequence Mining

Here, we discuss the proposed method for removing outlier behaviour in an event log. This method requires an input event log and some parameters that are adjustable by the user. It also consists of three main steps. In the first step, we discover sequential rules and sequential patterns from the event log according to the settings that are adjusted by the user. Then in the second step, based on discovered sequential rules and patterns for each trace we search if it consists of any outlier behaviour. Finally, we remove traces with outlier behaviour from the event log. The pseudo-code of the proposed algorithm is given in Algorithm 1.

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 patten with 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 and minimum confidence equal to 0). Then remove all sequential patterns with the support value of higher than $Supp_O$ (we could call them normal sequential patterns) from all sequential patterns. In an informal way we say that $OddPatterns = Patterns_{MinSupport=0} - Patterns_{MinSupport=Sup_O}$.

As it is mentioned 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. We discover high probable sequential rules in both directions. In other words, we discover both $A \rightarrow C$ 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 computation complexity of our proposed method. The complexity of discovering sequential patterns (in an event log) is $O(n^{L_P})$ and complexity of discovering sequential rules is $O(n^{L_A+L_C})$.

After discovering odd sequential patterns and high probable sequential rules, in the second step, we try to 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:

- Traces with antecedence of a high probable sequential rule, should also contain the consequence of that rule.
- Existence of odd sequential patterns considered as outlier behaviour.

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

In general, we use odd sequential patterns to detect added or wrong ordered activities in process instances. For example in Figure 1, for trace $\langle a, e, b, d, c, f, g, h\rangle$, we have the odd pattern $\langle e, f\rangle$. So, in this trace one of the *e* or *f* is not normal 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 Figure 1, for 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 occurs after *b* in this trace. Therefore, we consider outlier behaviour in this trace.

Finally, in the third step, all traces that contain such outlier behaviour will be removed from the event log. Therefore,there is no outlier behaviour in any trace of the filtered event log that will be returned back to the user.

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 $T \in L$ do
        for each Rule $R \in$ ProbableRules do
            if ($T$ contains Antecedence of $R$ & does not hold $R$ ) then
                OutlierFlag= 1
        for each Pattern $P \in$ OddPatterns do
            if ($T$ contains $P$ ) then
                OutlierFlag= 1
        if (OutlierFlag ! = 1) then
            FilteredLog ← $T$
    return (FilteredLog)
```

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]. This plug-in takes an event log as an input with some parameters and outputs a filtered event log. In this implementation, for discovering sequential rules we use the *rule growth* algorithm [5] and also used *prefixspan* [4] for sequential patterns whether it is both of them have been implemented in the `SPMF` tool [26].

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 of `RapidMiner` that combines scientific workflows with a range of (`ProM`-based) process mining algorithms [27].

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

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 a reference models as the ground truth that indicates which of the traces contain outlier and which ones are clean. In Table 2, we present the event logs that are used in this evaluation with some characteristics of them. 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.

Table 2: 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 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 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 [28]. 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

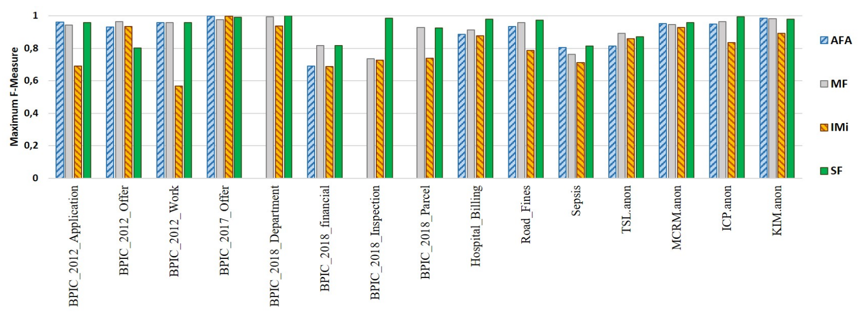
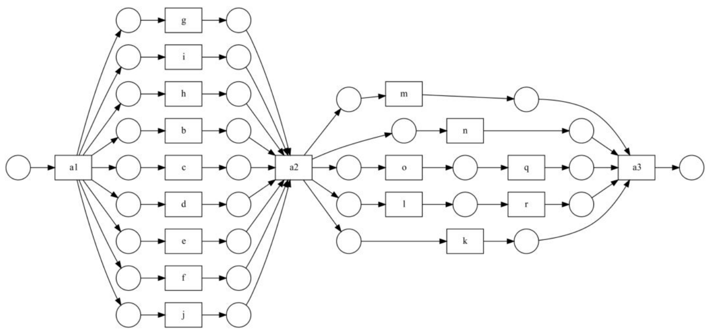


Fig. 4: The best F-Measure for using different filtering methods on some real event logs.

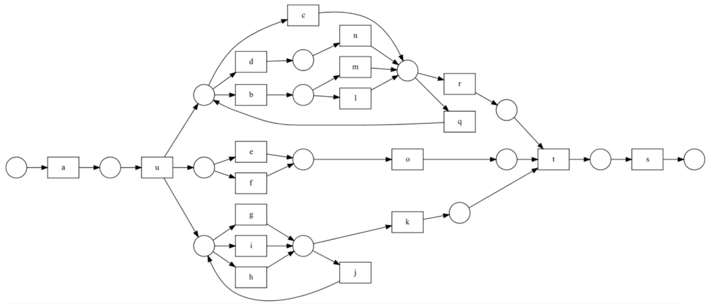
that combines fitness and precision: $\frac{2 \times Precision \times Fitness}{Precision + Fitness}$. Moreover, to assess outlier detectability of filtering methods, we map the problem to a classification problem. In other words, we should detect whether a trace contains outlier behaviour or whether it is clean. Therefore, we use the well-known classification measure *F1-score* [29] 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 performance 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 [15] 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 [11] with its embedded filtering mechanism with 50 different thresholds from 0 to 1.

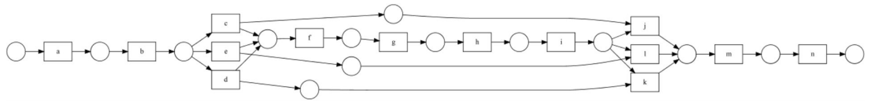
Figure 4, indicates the best F-Measures of discovered process models when different filtering methods are used to remove outlier behaviour. *AFA* and *MF* are consequently related to *Anomaly Free Automaton* [7] and *Matrix Filter* [6] methods and *SF* refers to the sequence mining based filtering method. We also compare these results with *IMi* that refers to using the Inductive Miner with its embedded filtering mechanism. Note that for 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. In general, an existence of long loops in the given event log has negative effects on the performance of *SF*. Also, in heavy presence of choice (*Xor*) 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 subsection 6.1. 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



(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.

characteristics of the event log. We found that if a rate of $1-\frac{Variant\#}{Trace\#}$ be lower it would be harder to discover a process model with high precision and fitness at the same time. Also, *AFA* is not able to filter $BPIC\_2018\_Parcel$ and $BPIC\_2018\_Inspection$. So, we have not corresponding results on Figure 4.

We also repeat this experiment for synthetic logs. The original process models of these synthetic event logs are presented in Figure 5. After generating corresponding event logs from these reference process models, we add different percentages of outlier behaviour to the original event logs. In this regard, we insert, remove and change the order of activities. The best *F-Measure* of applying different filtering methods on these event logs are given in Figure 6. For *High_Parallel* event log with increasing the percentage of inserted 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.

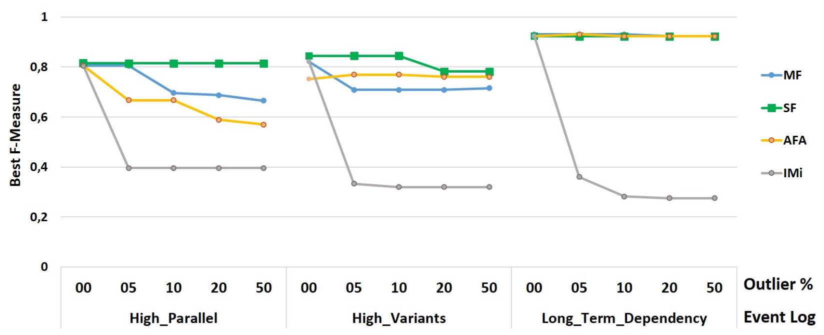


Fig. 6: The F-Measure of applying different filtering methods on synthetic event logs.

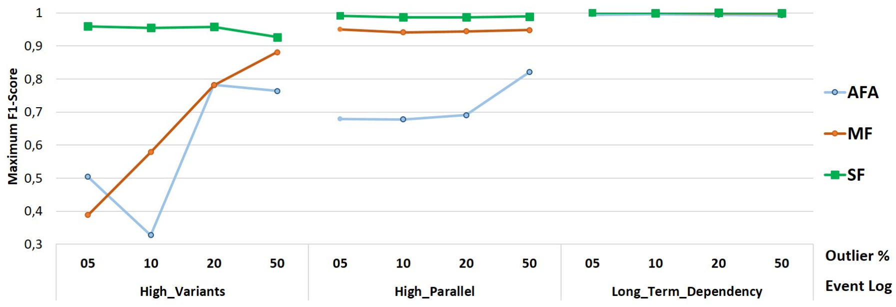


Fig. 7: The F1-Score of applying different filtering methods on synthetic event logs.

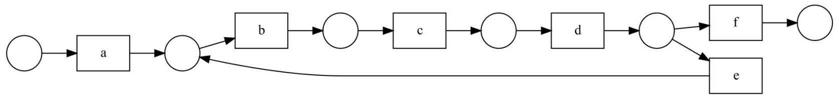


Fig. 8: A process model with loop behaviour.

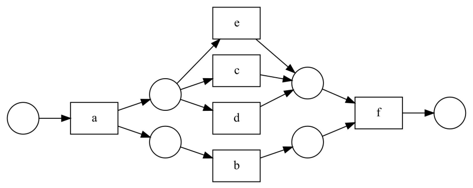


Fig. 9: A process model with *Xor* behaviour.

As shown in this experiment, even for 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 value of 1 even for the original process model and the original event log. As we use escaped edge definition [30] 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* of different filtering methods are indicated in Figure 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 Figure 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++ [12], we will discover a model exactly like Figure 5c that its *F-Measure* value will be equal to 1.

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.

6.1 Limitations

Our proposed filtering method uses sequential patterns and rules that cause to consider indirectly 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 Figure 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.

Another limitation of the proposed filtering method is that the presence of *Xor* behaviour has negative effects on the performance of it. For example, in Figure 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. 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 Figure 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 [6] and [7] 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 affords insights into the actual execution of business processes using available event data. Some of process mining algorithms are considered to work under the assumption that the input data is free of outlier behaviour. However, real event logs contain outlier (i.e., noise and infrequent behaviour) that typically 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 benefits from sequence mining algorithms to discover sequential patterns and rules. Using such information we able to discover flow relation of activities with 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 also offer it through 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 want to conduct larger experiments to find out in which situations a particular filtering mechanisms works better. We also want to estimate the filtering parameters based on features of the given event log.

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. 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) (2006) 67–87
4. Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: proceedings of the 17th international conference on data engineering. (2001) 215–224
5. Fournier-Viger, P., Wu, C.W., Tseng, V.S., Cao, L., Nkambou, R.: Mining partially-ordered sequential rules common to multiple sequences. IEEE Transactions on Knowledge and Data Engineering **27**(8) (2015) 2203–2216
6. Sani, M.F., van Zelst, S.J., van der Aalst, W. M. P.: Improving process discovery results by filtering outliers using conditional behavioural probabilities. (2017)
7. 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
8. 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)
9. 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)

10. Peterson, J.L.: Petri net theory and the modeling of systems. PrenticeHall, inc, Englewood Cliffst1 (1981)
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. Wen, L., van der Aalst, W. M. P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery **15**(2) (2007) 145–180
13. De Weerdt, J., vanden Broucke, S., Vanthienen, J., Baesens, B.: Active trace clustering for improved process discovery. IEEE Transactions on Knowledge and Data Engineering **25**(12) (2013) 2708–2720
14. Buijs, J.C., van Dongen, B.F., van der Aalst, W. M. P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: OTM, " On the Move to Meaningful Internet Systems", Springer (2012) 305–322
15. 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
16. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Avoiding Over-Fitting in ILP-Based Process Discovery. In: BPM. (2015) 163–171
17. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM). In: CIDM. (2011)
18. 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
19. 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
20. Gupta, M., Gao, J., Aggarwal, C.C., Han, J.: Outlier detection for temporal data: A survey. IEEE Transactions on Knowledge and Data Engineering **26**(9) (2014) 2250–2267
21. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning Structured Event Logs: A Graph Repair Approach. In: ICDE 2015. (2015) 30–41
22. Cheng, H.J., Kumar, A.: Process Mining on Noisy Logs —Can Log Sanitization Help to Improve Performance? Decision Support Systems **79** (2015) 138–149
23. van Zelst, S.J., Sani, M.F., Ostovar, A., Conforti, R., La Rosa, M.: Filtering Spurious Events from Event Streams of Business Processes. In: International Conference on Advanced Information Systems Engineering, Springer (2018) 35–52
24. Tax, N., Sidorova, N., van der Aalst, W.M.P.: Discovering more precise process models from event logs by filtering out chaotic activities. Journal of Intelligent Information Systems (2018) 1–33
25. Sani, M.F., van Zelst, S.J., van der Aalst, W. M. P.: Repairing Outlier Behaviour in Event Logs. In: International Conference on Business Information Systems. (2018) 115–131
26. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.: SPMF: a Java open-source pattern mining library. The Journal of Machine Learning Research **15**(1) (2014) 3389–3393
27. 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
28. 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,
29. Makhoul, J., Kubala, F., Schwartz, R., Weischedel, R., et al.: Performance measures for information extraction. In: Proceedings of DARPA broadcast news workshop, Herndon, VA (1999) 249–252
30. Munoz-Gama, J., Carmona, J.: Enhancing precision in process conformance: stability, confidence and severity. In: Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on, IEEE (2011) 184–191