Improving Process Discovery Results by Filtering Outliers using Conditional Behavioural Probabilities

Mohammadreza Fani Sani, Sebastiaan J. van Zelst, Wil M.P. van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{m.fani.sani,s.j.v.zelst,w.m.p.v.d.aalst}@tue.nl

Abstract. Process discovery, one of the key challenges in process mining, aims at discovering process models from process execution data stored in event logs. Most discovery algorithms assume that all data in an event log conform to correct execution of the process, and hence, incorporate all behaviour in their resulting process model. However, in real event logs, noise and irrelevant infrequent behaviour is often present. Incorporating such behaviour results in complex, incomprehensible process models concealing the correct and/or relevant behaviour of the underlying process. In this paper, we propose a novel general purpose filtering method that exploits observed conditional probabilities between sequences of activities. The method has been implemented in both the ProM toolkit and the RapidProM framework. We evaluate our approach using real and synthetic event data. The results show that the proposed method accurately removes irrelevant behaviour and, indeed, improves process discovery results.

Key words: Process Mining · Process Discovery · Noise Filtering · Outlier Detection · Conditional Probability

1 Introduction

Process mining is a research discipline that is positioned at the intersection of data driven methods like machine learning and data mining and Business Process Management (BPM) [1]. There are three types of process mining; *process discovery*, *conformance checking* and *process enhancement*. Process discovery aims at discovering process models from event logs. Conformance checking aims at assessing to what degree a process model and event log conform to one another in terms of behaviour. Finally, process enhancement aims at improving process model quality by enriching them with information gained from the event log.

Within process mining/process identification projects, process discovery is often used to quickly get insights regarding the process under study [1]. A business process analyst simply applies a process discovery algorithm on the extracted event log and analyzes its result. Most process discovery algorithms assume that event logs represent accurate behaviour. Hence, they are designed to incorporate all of the event log's behaviour in their resulting process model as much as possible.

Real event logs contain both *noise* and *infrequent behaviour* [2]. In general, noise refers to behaviour that does not conform to the process specification and/or its correct execution. Examples of noise are, amongst others, incomplete logging of process behaviour, duplicated logging of events and faulty execution of the process. Infrequent behaviour relates to behaviour that may occur, yet, in very exceptional cases of the process. For example, additional checks may be required when a loan request exceeds $\$10.000.000$. Incorporating noise and/or infrequent behaviour results in complex, incomprehensible process models concealing the correct and/or relevant behaviour of the underlying process. As such, when using process discovery for the purpose of process identification, we are often unable to gain any actionable knowledge by applying process discovery algorithms directly.

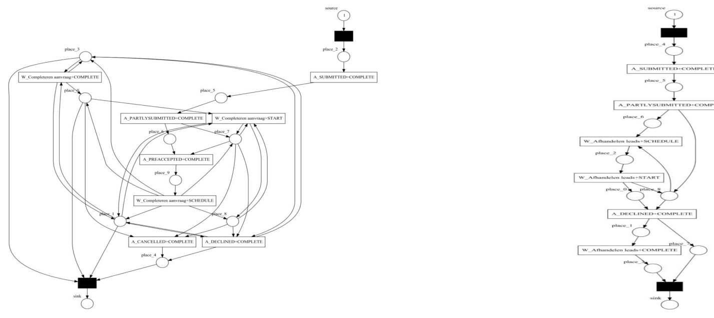
In this paper, we focus on improving process discovery results by applying general purpose event log filtering, i.e. filtering the event log prior to applying any arbitrary process discovery algorithm. Distinguishing between noise and infrequent behaviour is a challenging task and is outside the scope of this paper. Hence, we consider both noise and infrequent behaviour as *outliers* and aim at identifying and removing such outliers from event logs. We propose a generic filtering approach based on conditional probabilities between *sequences of activities*. The approach identifies whether certain activities are likely to happen based on a number of its preceding activities. Using the `ProM` (`http://promtools.org`) [3] based extension of `RapidMiner` (`http://rapidminer.com`), i.e. `RapidProM` [4], 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 outliers, and, as a consequence increases the overall quality of process discovery results. Additionally, we show that our method outperforms other general purpose process mining filtering techniques.

The remainder of this paper is structured as follows. Section 2 motivates the need for general purpose event log filtering methods. In Section 3, we discuss related work and after that, in Section 4, we explain our proposed method. Details of the evaluation and corresponding results are given in Section 5. Finally, Section 6 concludes the paper and presents future work in this domain.

2 Motivation

An interpretable process model helps business process analysts to understand what is going on in event data. However, often process discovery algorithms return results that are complicated and not understandable, because of outliers within the event logs used. Figure 1 illustrates how the application of filtering greatly reduces the complexity in a *real event log*, i.e. the event log of the *Business Process Intelligence Challenge 2012 [5] (BPIC 2012)*. Figure 1a shows a process model discovered using the ILP Miner of [6] for this event log, whereas Figure 1b shows the result of applying the same process discovery algorithm on 80% of the most frequent original behaviour.

In process mining, two quality measures are defined for measuring the behavioural quality of process models, i.e. fitness and precision [7]. Fitness computes how much behaviour in the event log is also described by the process model. On the other hand, precision measures the amount of behaviour described by the model that is also present in the event log. The fitness values of Figure 1a and Figure 1b are 0.57 and 0.46 whereas



(a) Result on whole event log.

(b) Result on 80% of event log.

Fig. 1: Process models discovered by applying the ILP Miner [6] on the BPIC 2012 log.

their precision values are 0.68 and 1.0 respectively. Thus, the model in Figure 1a describes more behaviour that is also present in the event log, however, in order to do this it greatly under-fits, i.e. it allows for much more behaviour compared to the model in Figure 1b. As a consequence, the model in Figure 1a is overly complex and ambiguous. The process model in Figure 1b, on the other hand, is much simpler while still covering at least 80% of the observed behaviour in the event log.

In our motivating example, by removing 20% of behaviour, we obtain a simpler model, i.e. Figure 1b that still accurately describes the underlying process. However, arbitrarily removing behaviour based on frequency is too ad-hoc and does not work when there is a lot of variety present within an event log, e.g. caused by parallelism. Therefore, we need more advanced filtering methods that take into account and exploit the actual behaviour described by the event log.

3 Related Work

In recent years, many process discovery algorithms have been proposed [8–13]. The first algorithms where designed to incorporate all behaviour in the event log [8, 12, 13]. More recently these algorithms have been extended to be able to handle outliers as well [14, 15]. However, these extended filtering techniques are tailored towards the internal working of the corresponding algorithm and hence do not work as a general purpose filtering technique. Other process discovery algorithms are specifically designed to cope with noisy and infrequent behaviour [9, 11]. However, these algorithms do not result in process models with clear execution semantics. Most of commercial process mining tools using these algorithms and their filtering are based on just the frequency of activities and their direct relations.

In this paper, we propose to *separate concerns*, and thus develop a novel, general purpose filtering technique that pre-processes event logs. In such way, any process discovery algorithm is able to benefit from effective identification and removal of outlier behaviour. In the remainder of this section, we focus on techniques developed for general purpose filtering in the process mining domain.

Table 1: Overview of filtering plugins in ProM.

Plug-in	***Applications***	***Main Method***
Filter Log using Simple Heuristics	Helpful for removing traces and activities based on frequency of events or the presence of certain start/end events.	Frequency/position of events
Filter Log on Event/Trace Attributes	Useful when we want to just keep events/traces with specific attribute values.	Attribute values
Dotted Chart	Allows us to visually select specific traces in event logs (usually base on a time frame).	Time window
Transition Systems Miner	Helpful to project traces/events on specific transitions and/or states.	Frequency of transitions
Filter Log using Prefix-Closed Language	Allows us to remove events from traces.	Rule based

The vast majority of process mining research has an accompanying implementation in the process mining toolkit ProM. Most work on general purpose event log filtering concerns ad-hoc filtering implementations within ProM. Many of these implementations are useful when we aim at using specific subsets of traces/events of an event log instead of the whole event log. In Table 1, the main filtering plugins are listed, accompanied by a brief description of their applications and methods. All plugins take an event log as an input and return a *filtered event log* as an output. Moreover, they need some form of domain knowledge to work properly. In addition, typically the user needs to set one or more (complex) settings. However, they do not support generic outlier detection, i.e. in cases where we possess no or little domain knowledge.

Little research has been done in the field of general purpose filtering. In [16] a graph-based outlier detection method is proposed to detect inaccurate data in an event log. In [17] a method is proposed that detects non-fitting behaviour based on a given reference model and then repairs the event log. As we want to improve process discovery results and, in general, we do not have a reference model, this method is not useful for general purpose filtering. In [18] the authors propose to provide training traces to the PRISM algorithm [19] which returns rules for detecting outliers. However, in real event logs, providing a set of training traces that cover all possible outliers is impractical.

The most relevant research in the area of general purpose log filtering is the work in [20]. The authors propose to construct an Anomaly Free Automaton (AFA) based on the whole event log and a given threshold. Subsequently, all events that do not fit the AFA are removed from the filtered event log. Filtering event logs using AFA indeed allows us to detect and remove noisy and/or infrequent behaviour. However, the technique does not allow us to detect all types of outliers like incomplete traces, i.e. traces that fit the AFA perfectly yet do not terminate properly. Incorporation of such behaviour can still lead to infeasible process discovery results.

Finally, separating outliers from event logs and focusing just on them rather than all behaviour also has been studied [21], however, a detailed treatment of outlier detection is outside the scope of this paper.

4 Filtering with Conditional Behavioural Probabilities

As indicated in Section 3, most filtering approaches are not suitable for process discovery because they need additional information like reference model or a set of outlier traces. Furthermore, the AFA filter, which is the most suitable general purpose event log filter, has trouble identifying irrelevant infrequent behaviour. Therefore, we present a general purpose filtering method that is able to deal with all types of outliers. The main purpose of the filter is to identify the likelihood of the occurrence of an activity, based on its surrounding behaviour, e.g. how likely is it that activity a follows the sequence of activities $\langle b, c\rangle$. To detect such likelihood it uses the conditional probability of activity occurrences, given a sequence of activities. As we just consider a sample of behaviour in the underlying process, i.e. an event log, all computed probabilities are an estimation of the behaviour that truly happened. Prior to presenting the filtering method, we present some basic notations used throughout the paper.

4.1 Basic Notation and Definitions

Given a set X, a multiset M over X is a function $M\colon X \rightarrow \mathbb{N}_{\geq 0}$. We write a multiset as $M = [e_1^{k_1}, e_2^{k_2}, ..., e_n^{k_n}]$, where for $1 \leq i \leq n$ we have $M(e_i) = k_i$ with $k_i \in \mathbb{N}_{>0}$. If $k_i = 1$, we omit its superscript, and if for some $e \in X$ we have $M(e) = 0$, we omit it from the multiset notation. Also, $M = [\,]$ is an empty multiset if $\forall e \in X$, $M(e) = 0$. 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 $\mathcal{M}$.

Let $\mathcal{A}$ denote the set of all possible activities and let $\mathcal{A}^*$ denote the set of all possible 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 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, $tl(\sigma, 0) = \epsilon$. Sequence $\sigma' = \langle a'_1, a'_2, ..., a'_k\rangle$ is a subsequence of sequence σ if and only 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, i.e. σ is a subsequence of itself.

Event logs describe sequences of executed business process activities, typically in context of some case, e.g. a customer or some order-id. The execution of an activity in context of a case is referred to as an *event*. Events are unique. A sequence of events is referred to as a *trace*. A trace projected onto the activities it describes is referred to as a *trace-variant*. Thus, it is possible that multiple traces describe the same trace-variant, i.e. sequence of activities, however, each trace contains different events. An example event log, adopted from [1], is presented in Table 2. Consider all activities related to *Case-id 1*. Sara *registers a request*, after which Ali *examines it thoroughly*. William *checks the ticket* after which Ava *examine causally* and *reject the request*.

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. $L \in \mathcal{M}(\mathcal{A}^*)$.*

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

Case-id	Activity	Resource	Time-stamp
...	...	...	...
1	*register request (a)*	*Sara*	*2017-04-08:08.10*
1	*examine thoroughly (b)*	*Ali*	*2017-04-08:09.17*
2	*register request (a)*	*Sara*	*2017-04-08:10.14*
2	*check ticket (d)*	*William*	*2017-04-08:10.23*
1	*check ticket (d)*	*William*	*2017-04-08:10.53*
2	*examine causally (b)*	*Ava*	*2017-04-08:11.13*
1	*reject request (h)*	*Ava*	*2017-04-08:13.05*
...	...	...	...

We abstract from the notion of events in Definition 1. Each $\sigma \in \overline{L}$ describes an observed *trace-variant* whereas $L(\sigma)$ describes its frequency.

Definition 2 (Subsequence Frequency). *Let L be an event log over a set of activities $\mathcal{A}$ and let $\sigma \in \mathcal{A}^*$. The subsequence frequency of σ w.r.t L, written as $freq(\sigma, L)$, denotes the number of times σ occurs as a subsequence of any trace present in L.*

Given a simple example event log $L_1 = [\langle a,b,c,d\rangle^5, \langle a,c,b,d\rangle^3]$, we have $freq(\langle a\rangle, L_1) = freq(\epsilon, L_1) = 8$, $freq(\langle a,b\rangle, L_1) = 5$, etc.

Definition 3 (Conditional Occurrence Probability). *Let L be an event log over a set of activities $\mathcal{A}$ and let $\sigma \in \mathcal{A}^*$ be a subsequence. Given some $a \in \mathcal{A}$, the conditional probability of occurrence of activity a, given σ and L, i.e. $COP(a, \sigma, L)$ is defined as:*

$$COP(a,\sigma,L) = \begin{cases} \frac{freq(\sigma \cdot \langle a\rangle, L)}{freq(\sigma, L)} & \textit{if } freq(\sigma, L) \neq 0 \\ 0 & \textit{otherwise} \end{cases}$$

Clearly, the value of any $COP(a, \sigma, L)$ is a real number in $[0, 1]$. A high value of $COP(a, \sigma, L)$ implies that after the occurrence of σ, it is very probable that activity a occurs. For example, $COP(a, \sigma, L) = 1$ implies that if σ occurs, a always happens directly after it. Based on the previously used simple event log, we have $COP(b, \langle a\rangle, L_1) = \frac{5}{8}$.

4.2 Outlier Detection

We aim to exploit conditional probabilities present within event logs for the purpose of filtering event logs. Conceptually, after a given subsequence, activities that have a particularly low COP-value are unlikely to have happened and therefore their occurrence may be seen as outlier. To account for dependencies between activities and previously occurred activities at larger distances, we compute COP-values for subsequences of increasing length.

In our proposed method, for each $i \in \{1, 2, ..., k\}$ we construct a COP*-Matrix*. Assume there are a total of m unique subsequences with length $1 \leq l \leq k$ in an event log. A COP-Matrix $\mathbf{A}^l_{COP}$ for length l is simply an $m \times |\mathcal{A}|$-matrix, where $\mathbf{A}^l_{COP}(\sigma, a) = COP(a, \sigma, L)$.

Algorithm 1 Outlier Detection Algorithm

```
procedure OUTLIERDTECTION($L$, $k$, $\kappa$)
Computing Probabilities:
    for ($l \in \{1, \dots, k\}$) do
        Build $\mathbf{A}^l_{COP}$, $\mathbf{A}^l_E$ and $\mathbf{A}^l_S$
    FilteredEventLog ← A new empty event log
Filtering:
    for ($\sigma \in L$) do
        Outlier ← $false$
        for ($l = 1 : k$) do
            for (subsequence $\sigma'$ with length $l$ and following activity $a$) do
                Find corresponding $COP(a, \sigma', L)$ in $\mathbf{A}^l_{COP}$, $\mathbf{A}^l_E$ and $\mathbf{A}^l_S$ values
                if ($\kappa > COP(a, \sigma', L)$) then
                    Outlier ← $true$
        if ($Outlier = false$) then
            FilteredEventLog ← Add $\sigma$ to FilteredEventLog
    return FilteredEventLog
```

We additionally compute conditional probabilities for *start* and *end* subsequences relatively. We let $\mathbf{A}^l_S$ denote a matrix describing the occurrence probability matrix of all subseqeunces $\sigma' = hd(\sigma)$ with $|\sigma'| = l$ for $\sigma \in L$. We are able to compute such probability by dividing the number of traces that start with σ' over the total number of traces in the log. Similarly we define $\mathbf{A}^l_E$ denote a matrix describing the conditional probability matrix of all subseqeunces $\sigma' = tl(\sigma)$ with $|\sigma'| = l$ for $\sigma \in L$ that is equal to $a = \epsilon$ in $\mathbf{A}^l_{COP}$. By doing so, we be able to handle outliers which occur in the start and the end parts of trace.

Given our different COP-Matrices, and a user-defined threshold κ, we identify each entry $\mathbf{A}^l(\sigma', a) < \kappa$ as an outlier. The pseudo-code of detecting outliers is present in Algorithm 1. In this fashion, it is possible to detect outliers that occur in start, middle or end part of traces. There are two ways to handle detected outliers. We are able to simply remove the corresponding event from the trace, i.e *event-level filtering*, or, remove the trace as a whole, i.e. *trace-level filtering*. However, removing an improbable event in a trace may make the trace to have more outlier behaviour. Hence, we just focus on *trace-level filtering*.

With increasing value of k (maximum length of subsequences), the complexity of the filtering method increases. The number of different strings we can generate over $\mathcal{A}$ with length k is $(|\mathcal{A}|)^k$ and total possible subsequences for some k: $\sum_{i=1}^{k} |\mathcal{A}|^i$ where $|\mathcal{A}|$ is the number of activities in the L. However, there is no need to compute COPs of all possible subsequences. For subsequences with length $k + 1$, it is sufficient to just consider $\sigma'.\langle a\rangle$ in level k that $\kappa \le COP(a, \sigma', L)$. For example, if at $k = 1$ $COP(c, \langle b\rangle) \le \kappa$, there is no need to consider $\langle b, c\rangle$ as a subsequent at $k = 2$, even though the $COP(a, \langle b, c\rangle)$ be higher than κ.

4.3 Implementation

To be able to combine the proposed filtering method with any process discovery algorithm, we implemented the *Matrix Filter* plugin (*MF*) in the `ProM` framework (`svn.win.tue.nl/repos/prom/Packages/LogFiltering`). The plugin takes an event log as an input and outputs a filtered event log. The user is able to specify threshold κ and whether event-level or trace-level filtering needs to be applied. The maximum subsequence length to be considered also needs to be specified.

In addition, to apply our proposed method on various event logs with different filtering thresholds and applying different process discovery algorithms with different parameters, we ported the *Matrix Filter* (*MF*) plugin to `RapidProM`. `RapidProM` is an extension of `RapidMiner` that combines scientific workflows [22] with a range of (`ProM`-based) process mining algorithms.

5 Evaluation

To evaluate the usefulness of filtering outliers using our method, we have conducted several experiments using both synthetic and real event data. The purpose of these experiments is to answer the following questions:

1. Does *MF* help process discovery algorithms to return more precise models?
2. How does the performance of *MF* compare to *AFA* filtering method?

To evaluate discovered process models, we use fitness and precision (introduced in Section 2). There is a trade off between these measures [23]. Sometimes, removing a little behaviour causes a decrease in fitness value, yet increases precision. To strike a balance between fitness and precision, we use the F-Measures metric that combines fitness and precision: $\frac{2 \times Precision \times Fitness}{Precision + Fitness}$. Also, filtering time and process model discovery time in milliseconds have been measured. Note that in all experiments, filtered event logs are only used in the process discovery part. Computing the F-Measure for all process models is done using the corresponding raw, unfiltered event logs. Furthermore, we only consider subsequences with length k in $[0, 2]$.

In the first experiment we investigate the effect of changing the κ in the *MF* threshold on the F-Measure w.r.t. different process discovery algorithms. We use the Inductive Miner [13] (IM) and the ILP Miner (ILP) [12]. Additionally we assess the interaction between our filtering technique and integrated filtering within the Inductive Miner, i.e. we use the IMi variant [14] with noise thresholds 0.1 and 0.3. We apply these algorithms and filtering methods on the BPIC2012 log. The results for this experiment are shown in Figure 2.

In this figure, each line corresponds to a discovery algorithm. The x-axis represents the threshold level of *MF*, the y-axis represents the corresponding F-Measure. Hence, for each technique, the data point $x = 0$ corresponds to not applying behavioural conditional probability filtering. We thus aim at finding out whether there exist regions of threshold values for which we are able to increase the F-measure when applying filtering. The F-measure of *IM* on this event log without using *MF* is 0.45. However, using the proposed filter increases the F-Measure of the discovered model to 0.80. Even for *IMi*, which uses an embedded filter, the *MF* increases the F-Measure from 0.69 and 0.7 to 0.81. As the ILP miner is more sensitive to outliers, *MF* helps more and its enhancement for this algorithm is higher. When increasing the threshold of *MF* to a value of 0.7 or higher, all the traces in the event log are removed and the fitness and F-Measure of the discovered model will equal to 0. The best result, i.e. an F-measure of 0.81, is achieved by IMi with threshold 0.1 and *MF* threshold of 0.09.

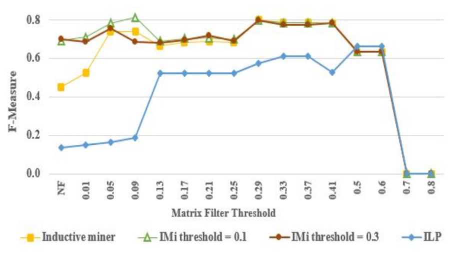


Fig. 2: Applying process discovery algorithms on the BPIC2012 log with different *MF* thresholds.

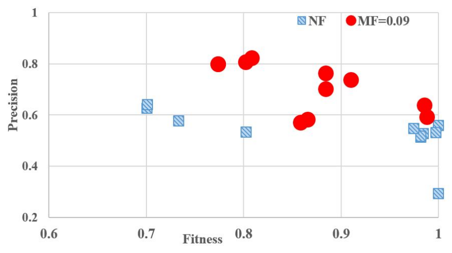


Fig. 3: Comparing process models discovered by 11 noise thresholds on the BPIC 2012 log with/without filtering.

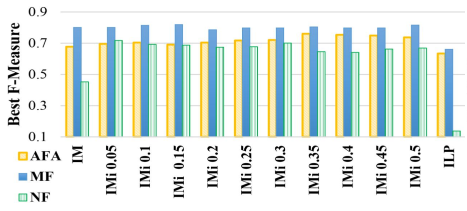
To illustrate the effect of filtering on the discovered process models, in Figure 3, we apply *IMi* with 11 internal thresholds ranging from 0.0 to 0.5 on the raw BPIC2012 and the filtered event log using *MF* with threshold 0.09. Here, each circle or square correspond to fitness and precision values related to one discovered model. A circle is related to applying *MF*, whereas squares relate to using the raw event log. As the results show, *MF* causes a little decrease in fitness value, yet yields an increase in precision value. The average of F-Measures when applying no filtering is 0.66 versus 0.77 in case of *MF* (with threshold 0.09). Thus, Figure 2 and Figure 3 indicate that *MF* improves process discovery results, i.e. the process models have an overall higher F-Measure.

In a second experiment, using the BPIC2012 and BPIC2017 [24] event log, we additionally assess what the maximal obtainable level of F-measure is for different process discovery algorithms, using different levels of internal filtering. We computed F-measures based on the unfiltered event log, and, maximized the F-measure result for both *MF* and *AFA*. With a workflow in the `RapidMiner`, for both filtering methods we filtered the event log using 40 different thresholds. The results are presented in Figure 4. This figure shows *MF* allows us to discover process models with higher F-Measures.

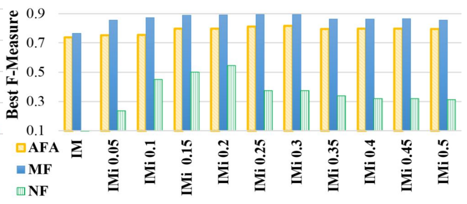
In Figure 5, we compare the average required time of applying the process discovery algorithms with/without filtering methods. In this figure, the y-axis represents the time in milliseconds with logarithmic scale. According to this figure, filtering methods reduce the required time for discovering process models, because there are fewer traces in the filtered event logs. Although, in *AF* the discovery time reduction is higher, the filtering time for this method is much higher than *MF* method. Therefore, in general *MF* seems to be faster than *AF*.

In the last experiment, to evaluate the ability of our proposed filtering method in detecting traces that contain outliers and corresponding effects on quality of process discovery algorithms, we use three synthetic event logs; *a12f0n*, *a22f0n* and *a32f0n*. These event logs are manipulated by adding niose with a probability of 0, 10, 20 or 50 percent [2]. The last two characters of event log indicate the probability of noise added to it, for example, *a22f0n20* correspond to *a22f0n* with 20% noise probability.

The noisy event logs are used for process discovery and the original synthetic event logs (free of noise) are used for computing the F-Measure. Similar to the experiment in Figure 3, the *IMi* algorithm with 11 various internal noise thresholds has been used. We



(a) BPIC 2012



(b) BPIC 2017

Fig. 4: Effect of filtering on best F-Measure of discovered models.

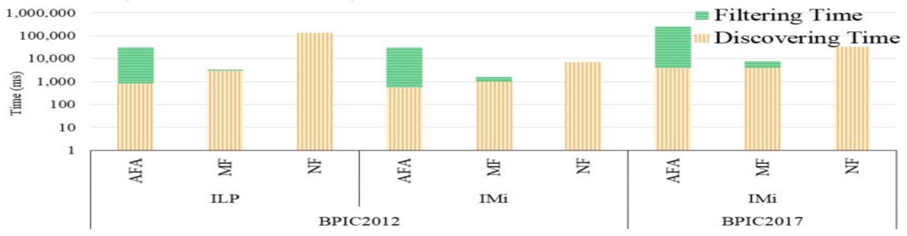
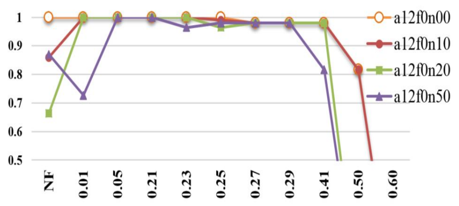
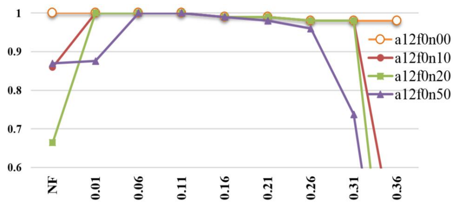


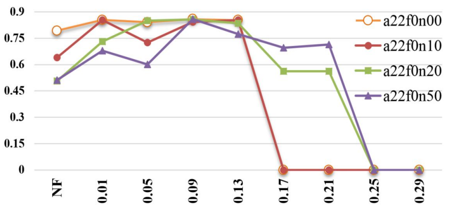
Fig. 5: Average of required time for process discovery with/without filtering



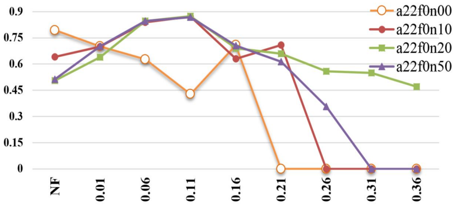
(a) *MF* on *a12f0n*



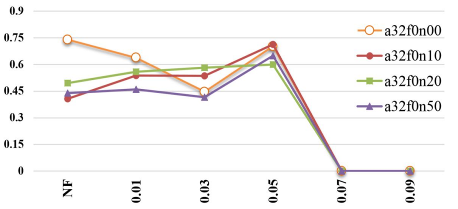
(b) *AFA* on *a12f0n*



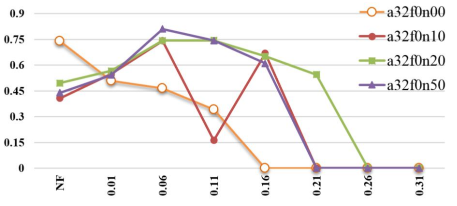
(c) *MF* on *a22f0n*



(d) *AFA* on *a22f0n*



(e) *MF* on *a32f0n*



(f) *AFA* on *a32f0n*

Fig. 6: Effect of filtering thresholds on F-Measures of synthetic event logs. *y*-axises are indicating values of best F-Measure and *x*-axises are showing the filtering thresholds.

only show results for the best F-Measure obtained. The results of this experiment are presented in Figure 6. According to this figure, the F-Measure of models improves when applying filtering methods. The improvement is observed to be much more substantial for event logs that contain more percentage of noise. For the *a12f0n* event log which has the simplest structure among these event logs, both methods lead to similar results. However, for *a22f0n*, applying *MF* results in better F-Measures. Finally, in *a32f0n* that corresponds to the most complex model with lots of parallelism, *AFA* performs better than *MF*. This can be explained by the fact that when a lot of paralelism is present, the conditional probability of non-outlier behaviour is low as well, i.e. parallelism implies a lot of variety in behaviour. In such situation it seems that using short subsequences (e.g. $k = 1$) or using a smaller κ value is a better choice for *MF*.

The experiments indicate that the proposed filtering method is useful for process discovery algorithms and allow us to obtain models with higher F-Measure whilst using a reduced discovery time. Hence, this shows that our method tends to outperform state-of-the-art process mining filtering techniques.

6 Conclusion

Process discovery is used to extract process models from event logs. However, real event logs contain noise and infrequent behaviour that hamper the direct applicability of existing process discovery algorithms. Separating such outliers from event logs is beneficial for process discovery techniques and helps to improve process discovery results.

To address this problem, we propose a filtering method that takes an event log as an input and returns a filtered event log based on a given threshold. It uses the conditional probability of the occurrence of an activity after a given sequence of activities. If this probability is lower than the given threshold, the activity is considered as an outlier.

To evaluate the proposed filtering method we developed a plugin in the `ProM` framework and the `RapidProM` extension of `RapidMiner`. As presented, we have applied this method on real event logs, and several process discovery algorithms. Additionally, we used the proposed method on three synthetic event logs. The results indicate that the proposed approach is able to help process discovery algorithms to discover models that strike a more adequate balance between different behavioural quality measures. Furthermore, using these experiments we show that our filtering method outperforms related state-of-the-art process mining filtering techniques.

We plan to evaluate the effect of using different values of k, i.e. length of subsequences. Also, other metrics besides the F-Measure like simplicity, generalization and structuredness could be analyzed. We want to apply event-level filtering and also assess different ways of using κ.

References

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

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) (2006) 67–87
3. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Rozinat, A., Verbeek, H.M.W., Weijters, A.J.M.M.: ProM: The Process Mining Toolkit. BPM (Demos) **489**(31) (2009)
4. 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)
5. van Dongen, B.: BPI Challenge 2012 (2012)
6. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering Relaxed Sound Workflow Nets using Integer Linear Programming. CoRR **abs/1703.06733** (2017)
7. 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: On the Move to Meaningful Internet Systems, OTM, Springer (2012) 305–322
8. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models From Event Logs. IEEE Trans. Knowl. Data Eng. **16**(9) (2004) 1128–1142
9. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM). In: CIDM. (2011)
10. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. Software & Systems Modeling **9**(1) (2008) 87–111
11. 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
12. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. Fundam. Inform. **94**(3-4) (2009) 387–412
13. 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
14. 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
15. 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
16. Ghionna, L., Greco, G., Guzzo, A., Pontieri, L.: Outlier Detection Techniques for Process Mining Applications. In: ISMIS 2008. (2008) 150–159
17. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning Structured Event Logs: A Graph Repair Approach. In: ICDE 2015. (2015) 30–41
18. Cheng, H.J., Kumar, A.: Process Mining on Noisy Logs —Can Log Sanitization Help to Improve Performance? Decision Support Systems **79** (2015) 138–149
19. Cendrowska, J.: PRISM: An Algorithm for Inducing Modular Rules. International Journal of Man-Machine Studies **27**(4) (1987) 349–370
20. 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
21. Yang, W., Hwang, S.: A Process-Mining Framework for the Detection of Healthcare Fraud and Abuse. Expert Syst. Appl. **31**(1) (2006) 56–68
22. Bolt, A., de Leoni, M., van der Aalst, W.M.P.: Scientific Workflows for Process Mining: Building Blocks, Scenarios, and Implementation. STTT **18**(6) (2016) 607–628
23. De Weerdt, J., , M., Vanthienen, J., Baesens, B.: A Robust F-measure for Evaluating Discovered Process Models. In: Proceedings of the CIDM. (2011) 148–155
24. van Dongen, B.: BPI Challenge 2017 (2017)