Cherry-Picking from Spaghetti: Multi-range Filtering of Event Logs

Maxim Vidgof(✉), Djordje Djurica, Saimir Bala, and Jan Mendling

Institute for Information Business, Vienna University of Economics and Business (WU), Vienna, Austria
maxim.vidgof@wu.ac.at

Abstract. Mining real-life event logs results into process models which provide little value to the process analyst without support for handling complexity. Filtering techniques are specifically helpful to tackle this problem. These techniques have been focusing on leaving out infrequent aspects of the process which are considered outliers. However, it is exactly in these outliers where it is possible to gather important insights on the process. This paper addresses this problem by defining multi-range filtering. Our technique not only allows to combine both frequent and non-frequent aspects of the process but it supports any user-defined intervals of frequency of activities and variants. We evaluate our approach through a prototype based on the PM4Py library and show the benefits in comparison to existing filtering techniques.

Keywords: Multi-range filter · Filtering event logs · Infrequent behavior · Process mining

1 Introduction

The goal of process mining is extracting actionable process knowledge using event logs of IT systems that are available in the organizations [1]. Process discovery is one of the areas of interest of process mining that is concerned with the extracting the process models from logs. With the development of process mining, a number of automated process discovery algorithms that address this problem has appeared.

The problem with automated process discovery of process models from event logs is that despite the variety of different algorithms, automated process discovery methods all suffer from joint deficiencies when used for real-life event logs [1]: they produce large spaghetti-like models and they produce models with either low level of fitness to the event log, or have low precision or generalization. Managing to correct these shortcomings proved to be a difficult task. Research by Augusto et al. [2] states that for complex event logs it is highly recommended to use filtering of the logs before automated process discovery techniques and that without this type of filtering precision of the resulting models is close to

S. Nurcan et al. (Eds.): BPMDS 2020/EMMSAD 2020, LNBIP 387, pp. 135–149, 2020.
https://doi.org/10.1007/978-3-030-49418-6_9

zero. The authors also highlight a research gap that is necessary to be closed suggesting the need to develop a filter which will can be tuned at will to deal with complex logs.

Therefore, the purpose of our study was to rectify this research gap by implementing a new filter, able to capture both most frequent behavior and the rare one. We created a prototype based on the PM4Py, process mining toolkit for Python [3]. Our prototype is fully customizable in which the user define an arbitrary number of ranges for both activities and variants of the process that user wants to analyze. In this research, we demonstrate how our technique helps to unveil new insights into the process using an illustrative example from the real-world event log.

This paper is structured as follows. Section 2 describes the problem setting and discusses common process mining techniques that rely on filtering of the logs in order to simplify models. Further, we present different types of filters and compare them. Finally, we derive requirements for new filter type. Section 3 presents a conceptual description of our filter with the formal definitions, while Sect. 4 presents an example that emphasizes the benefits of this technique. Section 5 shows the benefits of our technique against existing process mining tools. Section 6 concludes the paper and discusses future work.

2 Theoretical Background

This section describes the problem and provides an overview on related literature before deriving three requirements for a filtering technique.

2.1 Motivation and Problem Description

Data analysis plays a fundamental part in Business Process Management (BPM) and allows to improve processes based on facts. Process mining is the main technique to analyze processes using data which stem from event logs. These event logs keep track of the history of the various runs of the business process execute over time. Real world event logs typically contain a high number of cases, which may or may not differ from one another in the way they were handled. Mining such event logs usually results in models which contain an overwhelming amount of behavior (i.e., process variants). These models are also referred to as *Spaghetti* models as they make it hard to identify specific paths in their chaotic layout.

Spaghetti models provide little value as they are hard to understand. Literature has defined several techniques to overcome this problem, such as reducing complexity on a log level [11] and reducing complexity on a model level [6]. A main technique for reducing complexity offered by many of the process mining techniques is *filtering*. Usually process mining techniques show their results in visual interfaces which offer sliders to set up custom parameters for filtering. By moving these sliders the user are able to focus on specific aspects of the process.

What makes spaghetti models so complex is the fact that they show all possible behaviour, including paths that were seldom taken in the process. Therefore,

the focus of existing techniques from both academia and practice has been on filtering out this infrequent behaviour. We argue that in some cases, it is the infrequent behaviour that gives us better important insights on problems in the process, thus helping improvement. Indeed, existing tools such as ProM[1], Disco[2] and Celonis[3] are able to filter for specific behaviour. However, there is no way to set these filters in such a way that multiple variants or activities are shown together. This way of filtering leaves out important information, which might be seen for instance by a combination of the most and the least frequent cases.

Let us illustrate the problem through a running example. Figure 1 shows a simple complaint handling process adapted from [5]. The process works as follows. After a client files a complaint, (s)he immediately receives an automated confirmation message. Next, an employee brings the application to a meeting with colleagues in order to discuss a solution. The same employee is in charge of contacting back the customer with an apology and proposes a solution. The solution may be accepted or rejected by the client. In case of acceptance, the solution is executed right away. In case of rejection, the employee contacts the client to investigate on alternatives. As long as a reasonable alternative is found, the employee has a new meeting with colleagues to discuss the solution and proceed as usual. If no alternative solutions can be found, the complaint is brought to court and the process fails.

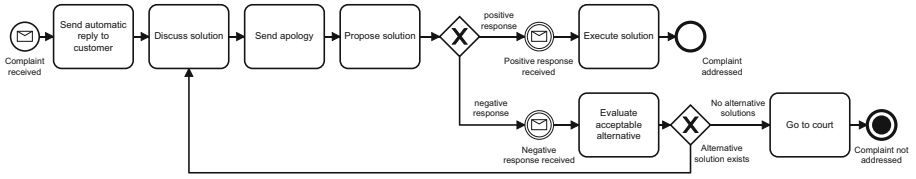


Fig. 1. Running example (adapted from [5])

There are several ways in which instances of the process may traverse the depicted process model. The *sunny case* scenario, is the one in which an agreement with the client is found right away. In a good process this case should occur frequently. On the opposite, the *rainy case* scenario consists of the cases which result in no agreement and the company is brought to court. In this case, the costs sustained from the company may be much higher than settling for a solution. An intermediate scenario is the one in which a customer does not accept the first proposed solution, but some iterations are done.

In order to improve the process, the company is interested to compare the sunny case scenario in order to understand which were the decisions and the proposed solutions that lead to the respective outcomes. Table 1a lists the activities involved in the process as well as their short labels for better readability.

[1] www.promtools.org.
[2] fluxicon.com/disco.
[3] www.celonis.com.

Typical process mining techniques retrieve variants as shows in Table 1b (i.e., sorted by frequency). Each variant represents one path in the output process model. In order to simplify the model, filtering must be used. However, if we filter out the least frequent behavior, we lose the information on the rainy case, which is the one that bears higher costs for the company. Some process mining tools like Celonis, would allow to select exactly the variant corresponding to the *rainy case* scenario. Unfortunately, this would leave out the *sunny case* scenario, which is also of interest of the company as this is the scenario in which the best decisions were taken.

Table 1. Process activities and variants

Activity or Event	Label
Complaint received	A
Send automatic reply to customer	B
Discuss solution	C
Send apology	D
Propose solution	E
Positive response received	F
Execute solution	G
Complaint addressed	H
Negative response received	I
Evaluate acceptable alternative	J
Go to court	K
Complaint not addressed	L

(a) Activities of the process

Variant	Count
⟨A,B,C,D,E,F,G,H ⟩	807
⟨A,B,C,D,E,I,J,C,D,E,F,G,H ⟩	132
⟨A,B,C,D,E,I,J,K,L ⟩	30
⟨A,B,C,D,E,I,J,C,D,E,I,J,C,D,E,F,G,H ⟩	21
⟨A,B,C,D,E,I,J,C,D,E,I,K,L ⟩	6
⟨A,B,C,D,E,I,J,C,D,E,I,J,C,D,E,I,J,C,D,E,F,G,H ⟩	2
⟨A,B,C,D,E,I,J,C,D,E,I,J,C,D,E,I,J,K,L ⟩	2

(b) Process variants ordered by trace frequency

The same consideration also holds for events and activities. Indeed, the company might be interested in activities or events which occur within a specific range of frequencies. For instance, the top 10 most frequent and the top 10 most infrequent activities can play a role into guiding process redesign. In other words, frequency of traces and activities do not necessarily reflect importance. There may be extremely infrequent variants or activities which have a very high impact on the process (e.g., Black Swans [10]). Hence, it is crucial that filtering does not compromise this information.

2.2 Filtering Techniques

According to Dumas et al. [5], process mining tools use two approaches to simplify event logs: *abstraction* and *event log filtering*. Abstraction is used to remove the subset of the nodes from the process map, producing a smaller dependency graph of the given event log. This way of simplifying process models is often beneficial because it enables model viewers to aggregate paths or activities of a given Spaghetti model and provides them with a better understanding about how the process functions on a macro level. However, while abstraction can visualize large event logs, it lacks the efficiency of coping with the full complexity of real-life event logs [6].

Consequently, process mining offers another type of event log simplification called event log filtering. Filtering an event log can be achieved with the use of three types of filters that remove a subset of the traces, events, or event pairs intending to produce a simpler log. *Event filters* allow users to remove or to keep all the events that satisfy a predefined condition set by the user. They allow users to focus only on a particular activity. *Event pair filter*, allow users to remove or keep all the pairs of events that fulfill a specific condition. This type of event log filtering is used to show a relation between two events and gather more insight into, for instance, situations where event A is followed by event B. Finally, using *trace filters* enables users to remove or retain all the traces from the log that fit the defined criteria. This filter can be used to, for example, show all the traces that occur with a defined level of frequency, or all traces that have a specific duration of cycle time [6].

In their paper on filtering out infrequent behaviour from event logs, Conforti et al. [4] mention more types of event filters mainly used in process mining tool ProM. First such filter is *Filter Log by Attribute* which removes all the events where the value of the attribute is not equal to the value defined by the user. It can also remove all the events that do not contain a certain selected attribute. Next, *Filter on Timeframe* serves to filter out all the events which fall into the desired timeframe. Some filters serve to filter out infrequent behaviour. One such instance is a *Filter Log using Simple Heuristics* which can remove all the traces that do not start and/or end with a particular event. It also can remove all the events related to the specific process task by calculating frequencies of event occurrence. Another example of the infrequent behaviour filter is *Filter Log using Prefix-Close Language*. This filter eliminates all the traces that are not a prefix of another prefix in the log by using a frequency threshold defined by the user.

While both abstraction and event log filtering techniques work well with structured processes but have problems visualizing and discovering less structured ones, recently, new techniques have been emerging that try to bridge this gap [9,12]. *Trace clustering* is a technique where the event log is divided into homogeneous subsets which are then used to create separate process models. This approach is able to cope with real flexible environments and improve process mining results. However, trace clustering is shown to suffer from a significant difference between clustering and the evaluation biases. The technique that tackles this problem, and manages to bridge this difference is *Active Trace Clustering* [12] inspired by principles of active learning. This approach borrows elements from machine learning and utilizes selective sampling strategy which enables an active learner to decide which instances to select based on their informativeness. Most frequently used informativeness measure is the frequency of the trace.

Several process discovery algorithms deal with noise in the logs are developed. The most well-known ones are *Heuristics Miner* [13], *Inductive Miner* [7], and *Fuzzy Miner* [6]. Heuristics miner deals with noise by introducing frequency-based metrics, while Inductive Miner uses two types of filters that accomplish this. The first filter applies a similar approach to Heuristics Miner and removes all the edges from the directly-follows graphs. In contrast, the second filter removes

edges that the first filter did not remove by using eventually-follows graphs. However, process models mined using Inductive miner are often oversimplified. A different approach to the previous two is Fuzzy Miner. This algorithm filters noise directly on the discovered model using the desired level of significance and correlation thresholds defined by users.

As we can see, there are numerous techniques and algorithms which can be used to simplify event logs and models to help users understand the core process better. However, all of them are achieving this by filtering out infrequent behaviour, considering it to be the noise in the event logs [1]. We argue that this is a substantial limitation that needs to be addressed since infrequent behaviour can carry important information which is lost by filtering it out of the log. For example, having an insight into rare cases can help companies detect errors in the process or even detect fraud. Furthermore, none of the presented techniques considers that users might want to observe a process model that comprises both the most frequent and infrequent traces of the process.

2.3 Requirements for a Filtering Technique

Against this background, we derive the following requirements for a filtering technique.

RQ1. (Select variants). A filtering technique must allow the user to *slice* the log. That is, it must offer a way of selecting process variants relevant to the user.
RQ2. (Select activities). A filtering technique must be able to *dice* the log. That is, it must offer a way of selecting the most relevant activities for the user.
RQ3. (Multi-range filtering). A filtering technique must be able to *slice* and *dice* on multiple ranges. That is, it must offer a way of selecting relevant information form several frequency intervals.

3 Technique for Multi-range Filtering

In this section we describe our filtering technique that allows to learn process models without ruling out infrequent behaviour. We show an overview of the technique, provide the necessary definitions and then describe the technique in detail.

3.1 Overview of the Technique

Our technique is summarized in Fig. 2. It takes as input an event log and two user defined multi-ranges. A multi-range is a set of intervals of frequencies. As we use frequencies, interval boundaries are from 0 to 1, where [0,0] means that we get the least frequent variant or activity, and [0,1] means that that we consider all possible behavior. The aforementioned multi-ranges are used respectively by

two filter types: *i)* variants filter; and *ii)* activities filter. These two filters can be used independently or consecutively. In the latter case, their application must follow the order: variants filter first. The output of each filter is a simplified event log, complying with filtering criteria. This event log can be used by any process mining technique to generate a process model which allows the user to analyze the data.

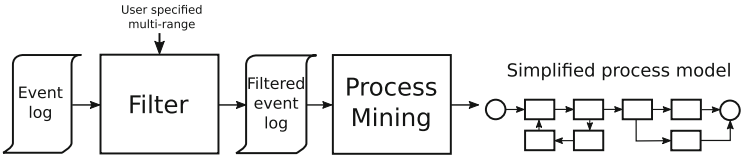


Fig. 2. Overview of the approach

3.2 Preliminaries

Definition 1 (Event, activity). *Let $\mathcal{A}$ be the universe of events. Each event has attributes. Let ANbe the set of attribute names. For any event $e \in \mathcal{A}$ and name $n \in AN$, $\#_n(e)$ is the value of the attribute n for event e. An activity is a specific attribute of an event, i.e., $\#_{activity}(e)$ is the activity associated to the event.*

For example, $\#_{activity}(e)$ = 'Discuss solution'.

Definition 2 (Trace, variant, event log). *A trace $t = \langle e_1, \ldots, e_n \rangle$ is a finite sequence of events. An event log $L \subseteq \{t\}^*$ is a multi-set of traces, i.e. A process variant is a subset of traces $V \subseteq L$. Variants group together traces which have similarities to one another and differences to traces in other variants.*

An example of trace is $t = \langle a, b, c, d, e, f, g, h \rangle$. An example of log is $L = [\langle a, b, c, d, e, f, g, h \rangle^{20}, \langle a, b, c, d, e, i, j, k, l \rangle^5]$. In this event log, the first trace occurs 20 times whereas the second one occurs 5 times.

Definition 3 (Variant frequency, Activity frequency). *Variant frequency $vf(V)$ is defined as the frequency occurrence of its constituting traces $t \in V$. Activity frequency $af(a)$ is defined as the sum of the number of times activity a in the event log L.*

For example, given $L = [\langle a, b, c, d, e, f, g, h \rangle^{20}, \langle a, b, c, d, e, i, j, k, l \rangle^5]$, then $vf(\langle a, b, c, d, e, f, g, h \rangle) = 20$ and $af(a) = 25$.

A filtering technique is a function $f : L \rightarrow L'$ which transforms an event log L into a simpler event log L'. Next, we use the given definitions to describe the algorithms used by our technique.

3.3 Implementation

Our implementation provides two filters: the variants filter and the activities filter. These two filters are composable but their application is not commutative, i.e. it has to be performed in strictly defined order. Namely, first the variants filter is applied and then the activities filter is applied on the results of the variants filter. In case the former one filtered out some variants, only the activities present in the remaining variants can be used in the latter one.

We are interested in filtering at multiple ranges in the event log. These ranges represent frequencies expressed by the user in the form of sets of intervals. That is, $R = \{[min_0, max_0], [min_1, max_1], \dots [min_n, max_n]\}$ with $min_i <= max_i$, $i = 1, \dots, n$ signifies that the user want to retain from the log an amount of information that falls into either of the intervals $[min_0, max_0], \dots, [min_n, max_n]$. Ranges can be applied to both filtering on the variants level - referred to as R_v - and filtering on the activities level - R_a. Since the range boundaries are specified as frequency percentages, the minimum value of min_i is 0, and the maximum value of max_i is 1. We also establish that $[min, max]$ means that the boundaries of the interval are included and (min, max) means the boundaries are excluded. With this definition we can express the non-overlaps condition on the ranges specified by the user as $\forall i, j \in [0...n] \Rightarrow [min_i, max_i] \cap [min_j, max_j] = \varnothing$. This is a precondition for applying both the activity and the variants filters. In other words, ranges may share boundaries but they must not overlap.

Our implementation consists of three main blocks. First, the ranges specified by the user for each of the applied filters are checked for overlaps. If the ranges are incorrect, an error is produced and the filtering is not applied.

Second, if the ranges are correct, the variants filter can be applied. The variants are filtered according to Algorithm 1.

Third, we can apply Algorithm 2 on the resulting log. First, it builds a list of activities sorted by their frequency, analogous to Algorithm 1. Then, a range filter is applied in the same manner. Finally, we iterate over all traces in the input log and rebuild them in such a way that only filtered activities remain in the trace. The new trace is appended to the output log only in case it is not empty, i.e. it contains at least one of the activities that should remain.

Algorithm 1. Filter variants

Input: Event log L. Ranges $V = \{(min_0, max_0) \dots (min_m, max_m)\}$, $m \in \mathbb{N}_0$
Result: A new event log $L' \subseteq L$

1 $variants \leftarrow \forall$ variants $\in L$;
2 $variants \leftarrow$ **sort** $variants$ **by** $vf_L(variant)$;
3 $nr\_variants \leftarrow |variants|$;
4 $indices \leftarrow \bigcup_{i=0}^{m} \{n \in \mathbb{N}_0 | n \in [round(min_i \times nr\_variants), round(max_i \times nr\_variants)]\}$;
5 $filtered\_variants \leftarrow \bigcup_{i \in indices} variants_i$;
6 $L' \leftarrow \forall trace \in L \cap filtered\_variants$;

Algorithm 2. Filter activities

Input: Event log L'. Ranges $A = \{(min_0, max_0), \ldots, (min_p, max_p)\}$, $p \in \mathbb{N}_0$
Result: A new event log $L'' \subseteq L'$

$activities \leftarrow dict(key = activity, value = af_L(activity))$;
forall $variant \in L'$ **do**
  forall $activity \in variant$ **do**
    if $activity \notin activities$ **then**
      $activities = activities \cup \{activity\}$;
      $af_L(activity) \leftarrow vf_L(variant)$;
    else
      $af_L(activity) \leftarrow af_L(activity) + vf_L(variant)$
    end
  end
end
$activities \leftarrow$ **sort** $activities$ **by** $af_L(activity)$;
$nr\_activities \leftarrow |activities|$;
$indices \leftarrow \bigcup_{i=0}^{p} \{n \in \mathbb{N}_0 | n \in [round(min_i \times nr\_activities), round(max_i \times nr\_activities)]\}$;
$filtered\_activities \leftarrow \bigcup_{i \in indices} activities_i$;
$L'' \leftarrow []$;
forall $trace \in L'$ **do**
  $new\_trace \leftarrow []$;
  forall $activity \in trace$ **do**
    if $activity \in filtered\_activities$ **then**
      $new\_trace = new\_trace \cup \{activity\}$;
    end
  end
  if $new\_trace \neq \varnothing$ **then**
    $L'' \leftarrow L'' \cup \{new\_trace\}$;
  end
end

4 Results

Next, we built a prototype to evaluate our technique. This section presents the results. First, we describe the experimental setup. Then we demonstrate that our technique addresses all the requirements by applying our technique to the running example we provided in Sect. 2.1. Last, we show the usefulness of our technique in a real-life log.

4.1 Experimental Setup

We implemented our technique as a prototype. We built our prototype using the PM4Py [3] library. It is a library for process mining implemented in the Python programming language. We used Jupyter notebook for our implementation. We tested on a laptop with Intel®Core ™ i7-8565U CPU @ 4.60 GHz x 4 machine with 16 GB of DDR4 RAM and Linux kernel 4.15.0-88-generic 64-bit version.

By default, our tool takes an event log in XES format as input but it can be also configured to accept event log in CSV format. The output is a filtered log, again, in XES or CSV format. The output of our tool can be used with any other process process mining tool. Apart from mining the resulting log in PM4Py, the user can export it and work on it with other tools like ProM, Disco, Celonis, etc. We used PM4Py and ProM in our evaluation. Our prototype is publicly available as open source software on GitHub[4].

4.2 Results on Artificial Log

We generated a log of our example process in Fig. 1 using BIMP[5]. The log contains 1000 cases and was built with the following rules: *i)* positive response is received with 80% probability; *ii)* negative response is received with 20% probability; *iii)* alternative solution exists with 80% probability; *iv)* no alternative solutions exist with 20% probability.

In order to evaluate our technique, let us apply our prototype on this artificial log. As already mentioned, the two filters can be used both separately and combined. First, we can use the variants filter to keep process behaviour that is of interest to us. Let us say, we are interested in the most frequent and the least frequent variants. To do that, we apply Algorithm 1 and specify two ranges for the filter: $R_v = \{[0, 0.15], [0.9, 1]\}$. It means we want to keep the 15% least frequent paths as well as 10% most frequent ones. It is very important to interpret these ranges correctly: by saying we take 15% most infrequent paths we do not mean taking 15% of the cases. Instead, we mean here paths that are between the 0th and the 15th percentile in a list of all variants in the input log sorted by their frequency.

We do not want to filter out any activities at this point, thus we specify one range $R_a = [0, 1]$ for the activities filter, meaning we want to keep 100% of activities. This gives us a filtered log L' that we can use further either in PM4Py or in any other tool. Figure 3 shows a Petri net resulting from applying Heuristics miner in ProM on the filtered log and adapted for better readability.

However, we may also want to filter activities at this point. Note that as we already applied the first filter on our log, only the activities present in the selected variants will be available for us to pick from. Let us say, we want to see the least frequent activities as well as the ones of medium frequency but not the most frequent ones. In order to do that, we can set multiple ranges for

[4] https://github.com/MaxVidgof/cherry-picker.
[5] http://bimp.cs.ut.ee.

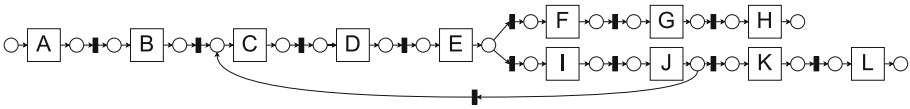


Fig. 3. Model from the artificial log in Table 1 with variants ranges $R_v = \{[0, 0.15], [0.9, 1]\}$ produced by heuristics miner and transformed into a Petri net.

the activities filter: $R_a = \{[0, 0.1], [0.1, 0.3], [0.4, 0.6]\}$. You can also see that the range boundaries are allowed to be the same but an overlap between ranges is not allowed.

Figure 4 shows the resulting model, again, adapted to improve readability. As we can see, it only includes the activities that are in the specified range: 40% least frequent activities and some activities with medium frequency. However, the new model does not contain the most frequent activities as they are outside of the specified range. This allows the user to concentrate on the less frequent and presumably more interesting activities.

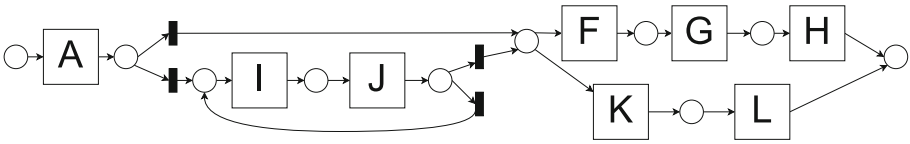


Fig. 4. Model from the artificial log in Table 1 with variants ranges $R_v = \{[0, 0.15], [0.9, 1]\}$ and activities ranges $R_a = \{[0, 0.1], [0.1, 0.3], [0.4, 0.6]\}$ produced by heuristics miner and transformed into a Petri net.

In conclusion, the proposed technique fulfills the requirements for an information-preserving filtering technique. More specifically, the requirements identified in Sect. 2.3 are addressed as follows.

RQ1. (Select variants) is addressed as the resulting model only shows the least frequent behaviour and the most frequent one.

RQ2. (Select activities) is addressed by the activities filter. Here the less frequent activities such as *Evaluate acceptable alternative* (J), *Go to court* (K) as well as the ones with medium frequency like *Complaint received* are present whereas the most frequent ones like *Discuss solution* (C) and *Send apology* (D) are filtered out.

RQ3. (Multi-range filtering) is addressed by our novel range specification approach. Instead of only selecting one threshold or manually picking some variants, the user can now specify multiple non-overlapping frequency ranges, and the union of sets of entities (variants or activities, depending on the filter) is written to the filtered log. The models above not only contain the least frequent traces like ⟨A,B,C,D,E,I,J,C,D,E,I,J,C,D,E,I,J,K,L⟩ but also the most frequent one such as ⟨A,B,C,D,E,F,G,H⟩. However, the traces with medium frequency are not included in the filtered log L'.

4.3 Results on Real-Life Logs

Next, we applied our technique on a real-life event log of sepsis cases [8]. This is a publicly available log containing more than 1000 traces and 15000 events, each trace corresponding to a pathway through the hospital.

By exploring the log, we can find out that there are 846 different variants, the most frequent of which includes only 35 cases that corresponds to slightly more than 3% of all traces in the log. There are also 784 variants having only a single conforming trace in the log. This means that the term frequent variant is not applicable to this log. Thus, it makes little sense to apply the variants filter on the log so we can set the range of the first filter to [0,1].

What is really of interest to us is the activities filter. While the filters of the traditional process mining tools only allow to keep the most frequent activities, which we will discuss in more detail in Sect. 5, our filter gives us more opportunities. Fir instance, we can decide to take a deeper look only into the least frequent activities. For this, we would set the activities filter to a range of [0, 0.25]. But we can also add additional ranges to these filter. Let us say, apart from the least frequent activities we are also interested in the one activity lying at the 65th percentile of frequency. This is also possible, for this we just set the second range to [0.65,0.65].

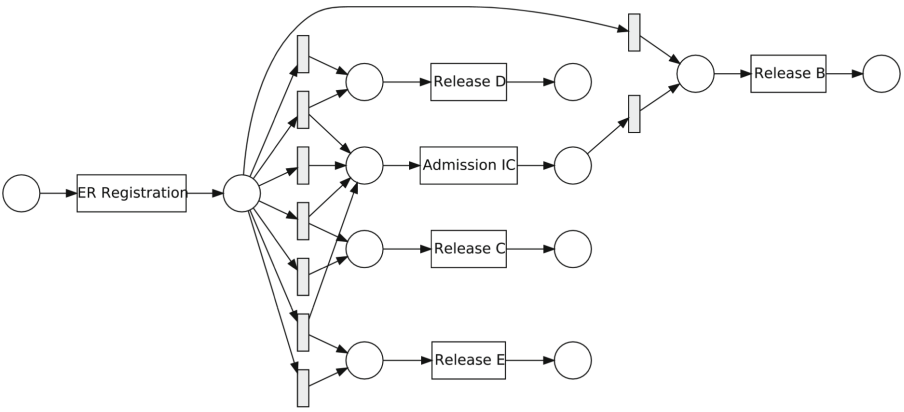


Fig. 5. Model from the real-life log with activities ranges [0,0.25] and [0.65,0.65] produced by heuristics miner and transformed into a Petri net

Now, if we apply the Heuristics miner on the filtered log and convert it to a Petri net, we will get a model in Fig. 5. Again, here we only see the activities that are in the specified range of frequency, and this picture cannot be achieved by any other process mining tool.

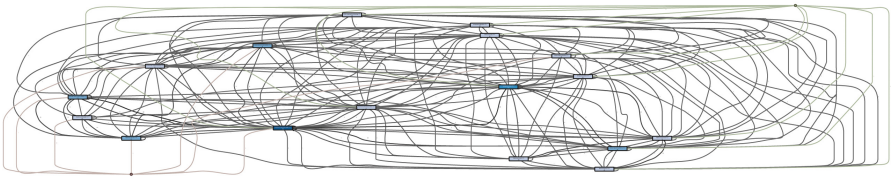
5 Discussion

Process mining allows the users to turn event logs into process models. However, real-life behaviour captured in these event logs of the process may be complex and exhibit notable variability. This leads to so-called spaghetti models (Fig. 6a) that are difficult to comprehend. Filtering reduces the complexity of such models by limiting the number of traces used to produce the model or the number of activities shown in the resulting model.

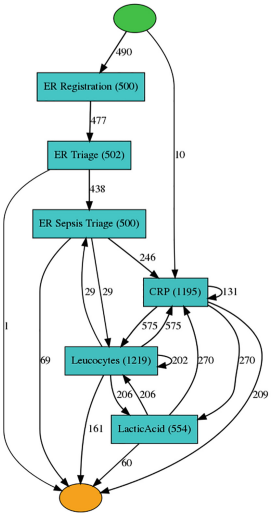
However, the users have little options to decide what information stays in the model and what can be left out for the moment, since existing process mining tools treat frequency as an ultimate measure of importance of a variant or an activity. Due to this, they only offer the user to keep the most frequent activities or paths. We claim, however, that a process can contain activities that are still very important despite infrequency but the tools provide virtually no possibility to include them and reduce complexity at the same time. Some of the tools provide the option to focus on any single path - also possibly an infrequent one - but then the big picture is lost and the process analyst has to manually incorporate this path in the model in case it is important. Moreover, no tool offers an option to focus on infrequent activities.

Our novel technique increases the utility of filtering event logs for the process analysts by allowing to set multiple ranges of frequency both for filtering variants and activities. Let us provide an illustrative example. Figure 6b shows a model produced by PM4Py heuristics miner from the real-life log about sepsis cases that we used in the previous section. Here, we used single-range filtering with $R_v = [0.65, 1]$ for the variants filter and $R_a = [0.6, 1]$ for the activities filter. Figure 6c is generated from a log where multi-range filtering was applied. In fact, only a slight modification was done to the activities filter: $R_a = \{[0.3, 0.3], [0.6, 1]\}$. This modification leads to the new activity *Return ER* - the patient returning to the hospital - appearing in the model. This activity, judging from the name, may be extremely important for the domain expert, although it does not happen frequently.

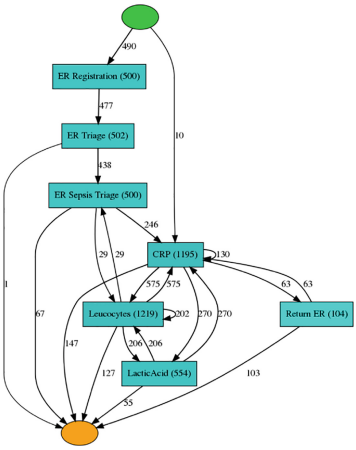
As this example shows, our filtering technique fills the gap that other techniques cannot fill. It does so by allowing the user to set multiple frequency ranges for both variants and activities, which in turn makes it possible to focus on previously disregarded behaviour and gain insights about the process behaviour that no other tool can provide. This can be beneficial in scenarios like monitoring of safety-critical processes, or controlling for possible fraudulent behaviour in companies. In such cases, it is of utmost importance that a filtering technique does not leave out information about potentially harmful cases.



(a) No filtering resulting in Spaghetti model



(b) single-range filtering



(c) multi-range filtering

Fig. 6. Impact of filtering on the resulting process models

6 Conclusion

In this paper we provide a novel filtering technique which sacrifices infrequent occurrence neither of process variants nor of process activities. We leveraged the PM4Py libraries to build a prototype which can work with multiple event logs formats. As well, the result of our technique can be input to several process mining algorithms. We tested our technique both on a synthetic log generated from a well known process model as well as with real-world event logs. Our evaluation shows that we can obtain new insights which were either too hard to implement or not offered by existing process mining tools.

Our work has limitations. At current stage, intervals are defined as lists of tuples in Jupyter notebooks. This does not target end users with limited programming skills. In future work, we plan to implement a user-friendly interface. Furthermore, we plan to apply our technique in real-world scenarios, such as auditing, in which multi-range filtering may unveil possible pattern of fraud or non-compliance. Finally, we plan to extend state of the art techniques by enriching them with multi-range capabilities.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Augusto, A., et al.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019)
3. Berti, A., van Zelst, S.J., van der Aalst, W.M.P.: Process mining for python (PM4PY): bridging the gap between process - and data science. CoRR abs/1905.06169 (2019)
4. Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. **29**(2), 300–314 (2017)
5. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, 2nd edn. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-642-33143-5
6. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_24
7. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
8. Mannhardt, F.: Eindhoven University of Technology. Dataset. Sepsis Cases - Event Log (2016). https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
9. Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace clustering in process mining. In: Ardagna, D., Mecella, M., Yang, J. (eds.) BPM 2008. LNBIP, vol. 17, pp. 109–120. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00328-8_11
10. Taleb, N.N.: The Black Swan: The Impact of the Highly Improbable, vol. 2. Random house, New York (2007)
11. Veiga, G.M., Ferreira, D.R.: Understanding spaghetti models with sequence clustering for ProM. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 92–103. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12186-9_10
12. Weerdt, J.D., vanden Broucke, S.K.L.M., Vanthienen, J., Baesens, B.: Active trace clustering for improved process discovery. IEEE Trans. Knowl. Data Eng. **25**(12), 2708–2720 (2013)
13. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM). In: CIDM, pp. 310–317. IEEE (2011)