International Journal of Cognitive Informatics and Natural Intelligence
Volume 14 • Issue 2 • April-June 2020

Filtering Infrequent Behavior in Business Process Discovery by Using the Minimum Expectation

Ying Huang, Gannan Normal University, Ganzhou, China
Liyun Zhong, Gannan Normal University, Ganzhou, China
Yan Chen, South China Agricultural University, Guangzhou, China

ABSTRACT

The aim of process discovery is to discover process models from the process execution data stored in event logs. In the era of "Big Data," one of the key challenges is to analyze the large amounts of collected data in meaningful and scalable ways. Most process discovery algorithms assume that all the data in an event log fully comply with the process execution specification, and the process event logs are no exception. However, real event logs contain large amounts of noise and data from irrelevant infrequent behavior. The infrequent behavior or noise has a negative influence on the process discovery procedure. This article presents a technique to remove infrequent behavior from event logs by calculating the minimum expectation of the process event log. The method was evaluated in detail, and the results showed that its application in existing process discovery algorithms significantly improves the quality of the discovered process models and that it scales well to large datasets.

KEYWORDS
Business Process, Infrequent Events, Minimum Expectation, Process Mining

1. INTRODUCTION

Process mining refers to a family of techniques in the field of process management used to support the analyses of business processes based on event logs. Business process mining aims at the automatic construction of models that explain the behavior observed in event logs (Van et al., 2007). There are three classes of process mining techniques: process discovery, conformance checking, and process enhancement. Process discovery is based on an event log, and a new model, an a priori model, is constructed or discovered based on the low-level events. Conformance checking is used when there is an a priori model. The existing model is compared with the process event log, and the discrepancies between the log and the model are analyzed. Performance mining is used when there is an a priori model. The model is extended with new performance information, such as the processing times, cycle times, waiting times, and costs, so that the goal is not to check for conformance, but rather to improve the performance of the existing model with respect to certain process performance measures.

DOI: 10.4018/IJCINI.2020040101

During the process mining/process identification procedure, process discovery is the first step to construct the prior module, and it is often used to quickly obtain insights into the process under study (Van et al., 2016). Most process discovery algorithms assume that the event logs represent the behavior accurately, and that the logs are clean. Thus, these algorithms are designed to incorporate all of the behaviors in the event log into their resulting process model as much as possible (Huang et al., 2018). However, real event logs contain outliers, and these outliers may represent noise or infrequent behaviors (Măruşter et al., 2006). In general, noise refers to behavior that does not conform to the process specification and/or its correct execution. Infrequent behavior relates to events that may happen in exceptional cases of the process (Sani et al., 2017). Previous works show that the low levels of infrequent behavior have a detrimental effect on the quality of the models produced by various discovery algorithms, such as the heuristics miner (Weijters et al., 2011), the Fodina process discovery (Vanden et al., 2017), and the inductive miner (Leemans et al., 2013) algorithms, even though these algorithms claim to have noise-tolerant capabilities.

This paper deals with the issue of discovering high-quality process models in the presence of infrequent behavior in the event logs, that is, by filtering the event log prior to applying any particular process discovery algorithm.

The remainder of this paper is structured as follows. In Section 2, we discuss the related work, and in Section 3, we define the proposed technique and explain our proposed method. Details of the evaluation and the corresponding results are given in Section 4. Finally, Section 5 concludes the paper and discusses future work.

2. RELATED WORK

A number of outlier detection algorithms have been proposed in the data mining field. These algorithms build a data model (e.g., a statistical, linear, or probabilistic model) that describes the normal behavior and considers all data points that deviate from this model as outliers (Aggarwal et al., 2015).

In the context of temporal data, these algorithms have been extensively surveyed by Gupta et al. (2014) (for events with continuous values, known as time series) and by Chandola et al. (2012) (for events with discrete values, known as discrete sequences).

According to Gupta et al. (2014), we can classify these approaches into three major groups. The first group encompasses the approaches that deal with the problem of determining if an entire sequence of events is an outlier. These approaches either build a model from the entire dataset, i.e., from all the sequences (e.g., Budalakoti et al., 2009, Florez-Larrahondo et al., 2005, Sun et al., 2006 and Zhang et al., 2003) or subdivide the dataset into overlapping windows and build a model for each window (e.g., Hofmeyr et al., 1998, Lane et al., 1997, 1999). While the approaches in this group can, in principle, be used for filtering out the infrequent process behavior in event logs, the filtering would be too coarse grained and lead to the removal of entire traces in the log, which would impact the accuracy of the discovered process model.

Approaches in the second group identify single data points as outliers (e.g., Basu et al., 2007, Keogh et al., 2002 and Muthukrishnan et al., 2004) or sequences thereof (e.g., Yankov et al., 2008) on the basis of a data model of the normal behavior in the log, e.g., a statistical model. These approaches are not suitable since they work at the level of a single time series. To apply them to our problem, we would need to treat the entire log as a unique time series, which would lead to mixing events from different traces based on their absolute order of occurrence in the log. Another option is to treat every trace as a separate time series.

However, given that the process events are not repeated often within a trace, their relative frequency would be very low, which would lead to considering almost all the events of a trace as outliers.

Finally, approaches in the third group identify the anomalous patterns within sequences (e.g., Gwadera et al., 2005 and Keogh et al., 2002). These approaches assign an anomaly score to a pattern based on the difference between the frequency of the pattern in the training dataset and the frequency of the pattern in the sequence under analysis. These approaches are not suitable in our case due to the absence of a noise-free training dataset that can be used as input.

The outlier detection algorithms, e.g., Gupta et al. (2011, 2014), that are given a graph built from the log as input identify the outlier subgraphs within the given graph. These approaches consider undirected graphs where the order dependency between the elements is irrelevant. While this filtering mechanism may work with process event logs, the removal of infrequent behavior would again be too coarse grained.

The execution of business processes supported by IT systems is generally recorded in most system or application logs. These logs can then be converted into event logs for process mining analysis. The event logs serve as the starting point for process mining. An event log is a multiset of traces. Each trace describes the life cycle of a particular case (i.e., a process instance) in terms of the activities executed. Each trace captures the footprint of a process instance in the log in the form of a sequence of events. Each event records the execution of a specific process task within a trace. Some process mining techniques assume that a log is complete and free from noise and may produce unsound models if this is not the case. However, real-life log data will contain a certain amount of noise.

To detect the processes within massive amounts of real-life log data, several noise-tolerant discovery algorithms have been proposed. The inductive miner algorithm is based on a divide-and-conquer approach that always results in sound process models (Buijs et al., 2012). This algorithm, using the directly follows dependency, generates the directly follows graph. The fuzzy miner algorithm (Conforti et al., 2017), another discovery algorithm, applies noise filtering posteriori directly on the discovered model. This algorithm is based on the concepts of correlation and significance and produces a fuzzy net where each node and edge are associated with a correlation and significance value.

The main purpose of the filter is to identify the likelihood of the occurrence of an activity based on its surrounding behavior because the process is composed of a sequence of activities, e.g., how likely it is that activity a follows a sequence of activities. The criterion for the event filter can be based on the expectation of activity occurrences, given a sequence of activities.

3. FILTERING WITH THE EXPECTATION

3.1. Basic Notation and Definitions

Definition 1 (Event log, event and trace): $\mathcal{L} = \left(\mathcal{E}, \mathfrak{I}, \mathcal{O}\right)$. Let $\mathcal{E}$ be the set of events, and $\mathfrak{I}$ be a finite set of tasks. There is a subjective function linking $\mathcal{E}$ to $\mathfrak{I}$. $\mathcal{E} \to \mathfrak{I}$. Here, $\mathcal{O} \subseteq \mathcal{E} \times \mathcal{E}$ .is a strict total ordering over the events. A (nonempty) sequence of events, $\mathrm{e} \in \mathcal{E}$ is a trace, τ . Thus, an event log, $\mathcal{L}$, is a multiset of traces.

The business process can be viewed as a directed connected graph, and the events in the process graph have a transitive closure, so we use the reachability matrix to show the transitive closure of a process graph. The criterion for the infrequent behavior detected can be based on the minimum expected for events, d_{minEU} :

$$EU_{i \to j} = \sum_{j=1}^{k} \left| n_i \right| * p_{i \to j} \tag{1}$$

$$p_{i \to j} = \frac{\left| e_i \to e_j \right|}{\left| e \right|} \tag{2}$$

$$d_{EU_{i\to j}} = \left\{\left(e_i, e_j\right) | \Sigma EU_{i\to j} < \zeta, 1 \le i \le n, 1 \le j \le n\right\} \tag{3}$$

where $|n_i|$ is the number of events i in a trace, which can be found in the reachability matrix, $|e_i \to e_j|$ is the number of edges from event i to event j. $|e|$ is the total number of edges in the trace and ζ is the threshold of the expectation value.

The interquartile range is often used to find the outliers in data, so we use the interquartile to obtain the ζ. First, we calculate the lower quartile (Q_1), second quartile (Q_2), and upper quartile (Q_3). Currently, we do not use a uniform method to obtain the quartile, here $Q_1 = 1 + (n-1)*0.25$, $Q_2 = 1 + (n-1)*0.5$, $Q_3 = 1 + (n-1)*0.75$. The interquartile range $\mathrm{IQR} = Q_3 - Q_1$, and the outliers here are defined as the observations that fall below the IQR.

Definition 2 (Infrequent events): The set of the infrequent events, $\hat{\mathcal{E}}$ is defined as $\hat{\mathcal{E}} = \left\{e_j \mid \min\left(d_{EU_{i\to j}}\right)\right\}$. Here, e_j is an event that has a nonzero minimum possibility of linking from event e_i to event e_j in $d_{EU_{i\to j}}$.

Definition 3 (Essential events): Given log data $\mathcal{L}$. the events that must be preserved during data filtering (i.e., deleting the infrequent events from the trace) are called the essential events.

The events that occurred are shown in Table 1. The corresponding trace graphs and their synthesized trace graphs are shown in Figures 1 and 2, respectively.

When the infrequent arcs and corresponding events are removed, the frequencies of the other arcs are changed, which affects the arc frequency distribution curve. To address this problem, we propose reiterating the log filtering process several times, using the filtered log as the input, until no more events are removed. During the filtering process, we must retain the connectivity of the

Table 1. Event log data

Trace1	<A,B,B,C,D,E>
Trace2	<A,B,C,D,C,E>
Trace3	<A,B,B,B,C,E>
Trace4	<A,B,C,E>

Figure 1. Log trace graph of the event data

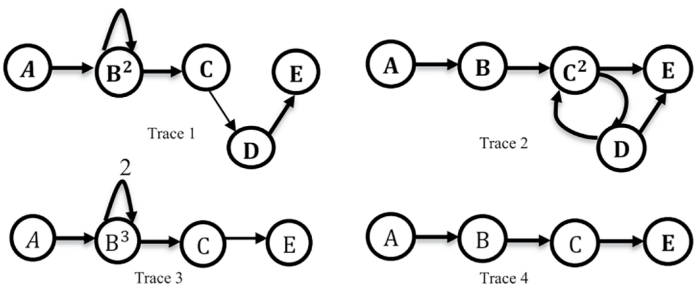
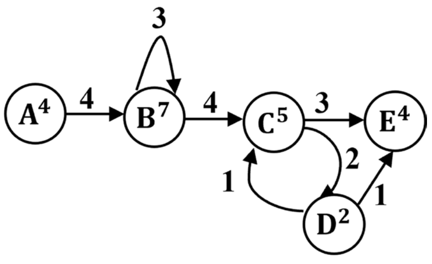


Figure 2. Synthesized trace of the event data



process. Certain events should be kept even at low frequencies. We can calculate the expectation of every event according to Definition 1:

$$\text{EU(A)} = 0 + 4\times\frac{4}{22} + 0 + 0 + 0 = \frac{16}{22}$$

$$\text{EU(B)} = 0 + 3\times\frac{3}{22} + 4\times\frac{4}{22} + 0 + 0 = \frac{25}{22}$$

$$\text{EU(C)} = 0 + 0 + 0 + 2\times\frac{2}{22} + 3\times\frac{3}{22} = \frac{13}{22}$$

$$\text{EU(D)} = 0 + 0 + 1\times\frac{1}{22} + 0 + 1\times\frac{1}{22} = \frac{2}{22}$$

$$\text{EU(E)} = 0 + 0 + 0 + 0 + 0 = 0$$

$$Q_1 = 1 + \left(n-1\right)*0.25 = 1 + 4*0.25 = 2$$

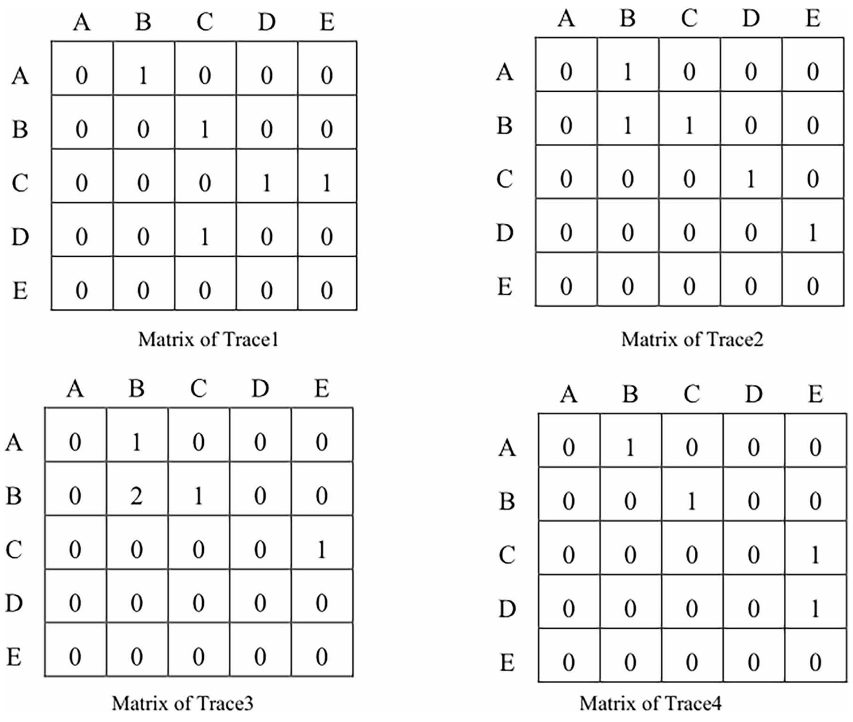
$$Q_3 = 1 + \left(n-1\right)*0.75 = 1 + 4*0.75 = 4$$

$$\text{IQR} = Q_3 - Q_1 = 4 - 2 = 2$$

Sorted from smallest to largest, the order of expectation is $\frac{2}{22}\ \frac{13}{22}\ \frac{16}{22}\ \frac{25}{22}$. Therefore, the second data point is $\frac{2}{22}$. We will filter events with an expectation less than or equal to $\frac{2}{22}$, unless the event is an essential event. Event *E* is the end event, so it must be kept. The expectation of event *D* is $\frac{2}{22}$, and the nonzero value indicates that the edge from the event to *D* to event *C* could probably be filtered; event *C* is not an essential event, so it will be deleted from the trace.

In Figure 1, the superscript of an event denotes the event frequency, and the number on the directed line is the frequency of two linked events. When the frequency is just one, we do not mark it. Utilizing the process mining algorithms, we can obtain the processes from the event data. The reachability matrix shows the relationship between events, where the matrix row is the outer link of the corresponding event, the matrix column is the inner link of the corresponding event, and the diagonal is the self-loop of the corresponding event. We can obtain the number of times the events have occurred by adding the numbers in the matrix row and the numbers in the matrix column. We can easily obtain the event relationships in different traces from Figure 3 and the sum of matrices in Figure 3. Figure 4 is the matrix of the synthesized trace graphs in Figure 1.

Figure 3. Matrix of the event data



	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	0	0
C	0	0	0	1	1
D	0	0	1	0	0
E	0	0	0	0	0

Matrix of Trace1

	A	B	C	D	E
A	0	1	0	0	0
B	0	1	1	0	0
C	0	0	0	1	0
D	0	0	0	0	1
E	0	0	0	0	0

Matrix of Trace2

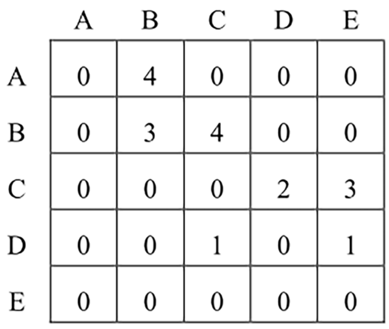
	A	B	C	D	E
A	0	1	0	0	0
B	0	2	1	0	0
C	0	0	0	0	1
D	0	0	0	0	0
E	0	0	0	0	0

Matrix of Trace3

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	0	0
C	0	0	0	0	1
D	0	0	0	0	1
E	0	0	0	0	0

Matrix of Trace4

Figure 4. Matrix of the events in Figure 2



	A	B	C	D	E
A	0	4	0	0	0
B	0	3	4	0	0
C	0	0	0	2	3
D	0	0	1	0	1
E	0	0	0	0	0

3.2. Infrequent Events Detected by the Minimum Exception

When d_{EU} is below the threshold value, according to Formula (2), it is very possible to detect an infrequent event, e_i. However, in special cases, such as start and end events, they are essential events that must remain regardless of the threshold value for these kinds of events.

Considering the example in Figure 3 and using a threshold value of 0.3, $\text{Min}\left(EU_{D\to j}\right) = \left\{EU_{D\to C}, EU_{D\to E}\right\} = 0.22 < 0.3$, meaning that the expectation from event *D* to the other events is below the threshold and $\hat{\mathcal{E}} = \left\{e_C, e_E\right\}$. However, *E* is an essential event, and it must remain. The link between *D* and *C* can be deleted, and the matrix is changed, as shown in Figure 6, and Figure 2 changes to Figure 7. The new trace 2 in the log was changed, as shown in Table 2.

Figure 5. Change of the trace 2 matrix

$$\begin{array}{c|c|c|c|c|c|} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 0 & 0 \\ \hline B & 0 & 0 & 1 & 0 & 0 \\ \hline C & 0 & 0 & 0 & 1 & 1 \\ \hline D & 0 & 0 & 1 & 0 & 0 \\ \hline E & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \Rightarrow \begin{array}{c|c|c|c|c|c|} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 0 & 0 \\ \hline B & 0 & 0 & 1 & 0 & 0 \\ \hline C & 0 & 0 & 0 & 1 & 1 \\ \hline D & 0 & 0 & \mathit{0} & 0 & 0 \\ \hline E & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Figure 6. Matrix of the events after filtering

$$\begin{array}{c|c|c|c|c|c|} & A & B & C & D & E \\ \hline A & 0 & 4 & 0 & 0 & 0 \\ \hline B & 0 & 3 & 4 & 0 & 0 \\ \hline C & 0 & 0 & 0 & 2 & 3 \\ \hline D & 0 & 0 & 0 & 0 & 1 \\ \hline E & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Figure 7. Synthesized trace after filtering

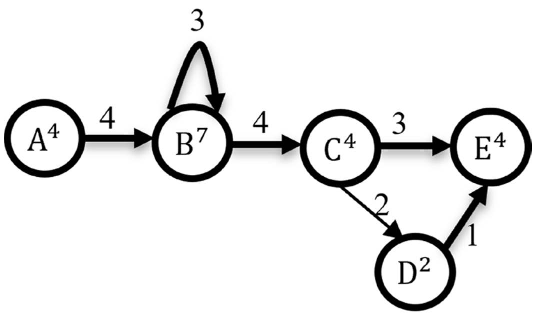


Table 2. Filtered event log

Trace1	<A, B, B, C, D, E>
Trace2	*<A, B, C, D, E>*
Trace3	<A, B, B, B, C, E>
Trace4	<A, B, C, E>

The process graph must be a simply connected graph, so removing a link between events can cause an unconnected graph, which must be avoided. Therefore, the events in a link are also known as essential events.

3.3. ILP Formulation of the Process

The log filter can be regarded as having an integer linear programming (ILP) format because there are only two types of tasks in the logs: kept (1) or deleted (0). This fits the classical ILP problem formulation of whether a selection is made or not. Thus, we can use an ILP approach to effectively judge which path is a minimum process in the log data. We introduce the following definitions before the ILP formulation:

- For events i and j, there exists $\mathcal{L}_{i,j} \in \{0,1\}$ in the simply connected process graph. When a subpath exists between events i and j, the result of the ILP problem is $\mathcal{L}_{i,j} = 1$, otherwise, $\mathcal{L}_{i,j} = 0$;
- For event $\mathrm{i} \in \mathcal{E}$, there exists $\mathcal{L}_s \in \{0,1\}$ in the simply connected process graph. When a subpath exists between the start event and event i, the result of the ILP problem is $\mathcal{L}_s = 1$, otherwise, $\mathcal{L}_s = 0$;
- For event $\mathrm{i} \in \mathcal{E}$, there exists $\mathcal{L}_e \in \{0,1\}$ in the simply connected process graph. When a subpath exists between event i and the end event, the result of the ILP problem is $\mathcal{L}_e = 1$, otherwise, $\mathcal{L}_e = 0$;
- For events $i, j \in \mathrm{E}$, there exists $\mathcal{L}_{s(i,j)} \in \{0,1\}$ in the simply connected process graph. When there are exit sub-paths from the start event through event i and event j, the result of the ILP problem is $\mathcal{L}_{s(i,j)} = 1$, otherwise, $\mathcal{L}_{s(i,j)} = 0$;
- For event $\mathrm{i}, \mathrm{j} \in \mathcal{E}$, there exists $\mathcal{L}_{e(i,j)} \in \{0,1\}$ in the simply connected process graph. When there are exit sub-paths through event i and event j that reach the end event, the result of the ILP problem is $\mathcal{L}_{e(i,j)} = 1$, otherwise, $\mathcal{L}_{e(i,j)} = 0$.

The ILP problem aims at minimizing the number of paths between the events:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} \left(e_i, e_j\right)$$

where there are paths between event e_i *and* e_j.

In frequent processes, the frequent events should satisfy the following constraints:

- For random events i and j, if they are connected:

$$C_{i,j} = 1 \tag{4}$$

- Every event i is reachable from the start events:

$$C_{s,i} = 1 \tag{5}$$

- Every event i is reachable to the end events:

$$C_{i,e} = 1 \tag{6}$$

- Every start event is reachable to the end events:

$$C_{s,e} = 1 \tag{7}$$

- For each event path, if events i and j are connected, event i is reachable from the start events:

$$\mathcal{L}_{s(i,j)} = 1 \Leftrightarrow C_{s,i} + C_{i,j} = 2 \tag{8}$$

- The constraints of the above formulas can be written as equivalent inequalities, shown as follows:

$$C_{s,i} + C_{i,j} - 2 \cdot \mathcal{L}_{s(i,j)} \geq 0 \tag{9}$$

$$C_{s,i} + C_{i,j} - 2 \cdot \mathcal{L}_{s(i,j)} \leq 1 \tag{10}$$

- For each event path, if event i and event j are connected, event i is reachable from the start event:

$$\mathcal{L}_{e(i,j)} = 1 \cdot C_{i,e} + C_{i,j} = 2 \tag{11}$$

- The constraints of the above formulas can be written as equivalent inequalities, shown as follows:

$$C_{i,e} + C_{i,j} - 2 \cdot \mathcal{L}_{e(i,j)} \geq 0 \tag{12}$$

$$C_{i,e} + C_{i,j} - 2 \cdot \mathcal{L}_{e(i,j)} \leq 1 \tag{13}$$

As we know, any event i must involve at least one path, which is from the start event via *i* to the end event.

The constraints can be written as equivalent inequalities as follows:

$\sum_{i,j=1}^{n} C_{i,j} \geq 1$ *if i = j,* indicates a self-loop

$$\sum_{i,j=1}^{n} C_{i,j} - N \cdot C_{s,i} \geq 1 \tag{14}$$

$$\sum_{i,j=1}^{n} C_{i,j} - M \cdot C_{i,e} \geq 1 \tag{15}$$

where *N* is the number of paths from the start event to event i, and *M* is the number of paths from event i to the end event.

There is at least one path from the start events though event i and event j in the process. At the same time, there is at least one path from events *i* and *j* to the end events in the process.

The infrequent event filtering methods presented above are described in pseudocode in Algorithm 1.

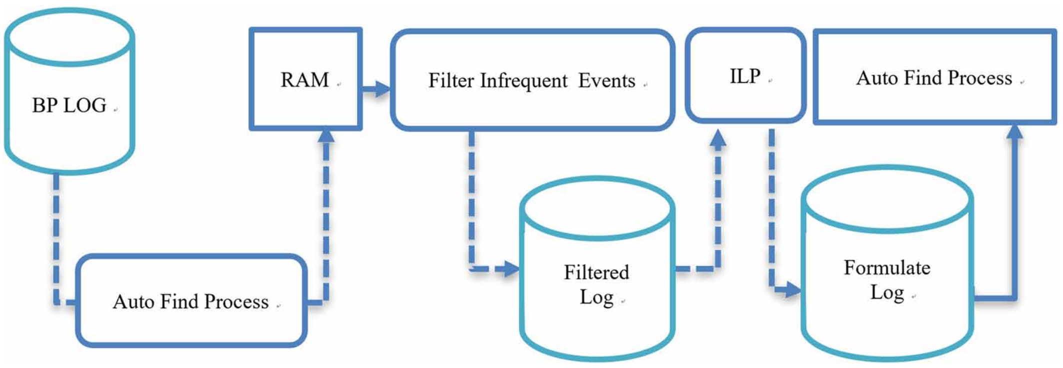
4. EXPERIMENTS AND EVALUATION

In this section, we present the results of the experiments to assess the advantage of our event filtering method. The design of the experiment is illustrated in Figure 8.

Algorithm 1. Infrequent event detection

```
Input: Event log $\mathcal{L}$
1    Original trace $\Gamma$ = Auto Find Process ( $\mathcal{L}$ )
2    Relation Ability Matrix RAM( $\mu$ ) $\Leftarrow$ Compute RA $(\Gamma)$
3    Sorted Expectation SE( $e_i, e_j$ ) $\Leftarrow$ Compute Expectation (RAM)
4    Filtering Threshold $\zeta$ $\Leftarrow$ Interquartile Range (SE)
5    Infrequent Event IFE( $e_j$ ) $\Leftarrow$ SE( $e_i, e_j$ ) $< \zeta$
6    Aligned( $\mathcal{L}'$ ) $\Leftarrow$ Delete Infrequent Event( $e_j, \mathcal{L}$ )
7    $\mathcal{L}''$ $\Leftarrow$ ILP( $\mathcal{L}'$ )
8    Original trace $\Gamma'$ $\Leftarrow$ Auto Find Process ( $\mathcal{L}''$ )
```

Figure 8. Framework for the experiments



In process mining, two quality measures are defined for measuring the behavioral quality of the process models: the recall and precision metrics (Buijs et al., 2012). The recall metric computes how much behavior in the event log is also described by the process model. On the other hand, the precision metric measures the amount of behavior described by the model that is also present in the event log. The precision and recall metrics counter each other, so that increasing one of them reduces the other. A metric that combines precision and recall is the harmonic mean of precision and recall, which is the traditional F-measure or balanced F-score:

$$\text{F-score} = 2 * \frac{precision * recall}{precision + recall} \tag{16}$$

4.1. Experimental Design and Datasets

For the experiments, we use three real-life logs from different domains and of different sizes, and we do not have insights into the types of infrequent behavior. We used these logs to evaluate the generalizability of the results obtained with the first two experiments. The three logs are publicly available: the hospital log, sepsis cases and hospital billing[1].

The first dataset is a real-life log of a Dutch academic hospital, and this log contains 1143 traces and 150291 events.

The sepsis cases event log contains events of sepsis cases from a hospital. The events were recorded by the ERP system of the hospital. Moreover, 39 data attributes are recorded containing 1050 traces and 15214 events.

The hospital billing event log was obtained from the financial modules of the ERP system of a regional hospital. The event log provided by the hospital contains events that are related to the billing of medical services. This log contains 100000 traces and 451359 events.

Conforti et al. emphasized that when analyzing events with filter measures, we need to ensure that the highest level of infrequent behavior is below 40% (Conforti et al., 2017). If the levels of infrequent behavior are above 40%, there is a contradiction since there should be a low proportion of infrequent behavior.

4.2. Results

Two of the methods deal with the removal of infrequent behavior: filtering the log using simple heuristics (*SLF*) and filtering the log using a prefix-closed language (*PCL*). *SLF* removes the traces that do not start or end with a specific event and events that refer to a specific process task, based on their frequency. *PCL* removes events from traces to obtain a log that can be expressed via a prefix-closed language. We compared our technique with the *SLF* and the *PCL* techniques using the above real-life data. Table 3 shows the change in the traces and events after using *SLF*, *PCL* and our technique to filter them.

From the aspect of the trace and event compression, our technique can match the SLF for the hospital billing and sepsis case logs. The PCL has a much higher compression than the other methods for the hospital and sepsis case logs; however, the excessive compression ratio might miss many essential events, and the event compression ratio even reached 92.8% for the hospital log, which is much higher than the upper limit of 40% (Conforti et al., 2017).

Figures 9 to 11 show the comparison results of the fitness, precision, and F-score, which were obtained using the three real-life logs when applying the three process mining techniques (heuristics miner, inductive miner and fuzzy miner). We used the default parameters for these process mining methods.

In Figures 9, 10 and 11, we used the filtered log and original log to test our technique. In Figure 9, the precision is improved, despite a reduction in recall. Therefore, the F-score improved for the

Table 3. Traces and events after filtering

	Original	SLF	PCL	OURS
Hospital Billing				
Traces	100000	83950	83109	83942
Events	451359	351296	354812	351184
Hospital				
40	1143	772	189	793
Events	150291	99781	10680	10327
Sepsis Cases				
Traces	1050	829	276	796
Events	15214	10090	2212	9748

Figure 9. Real-life log comparison of the heuristics miner algorithm

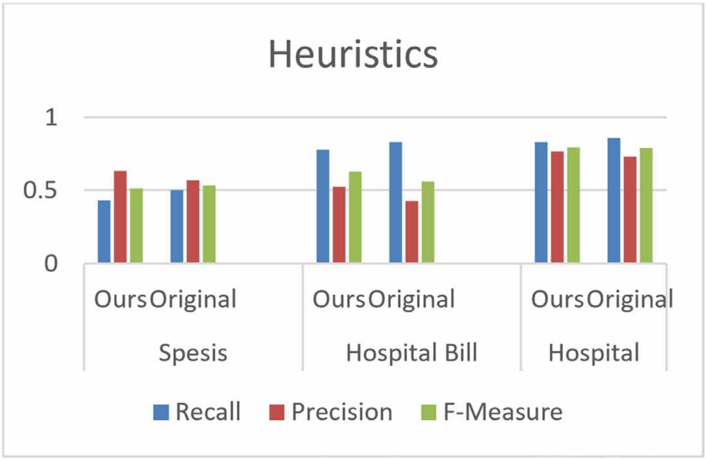


Figure 10. Real-life log comparison of the inductive miner algorithm

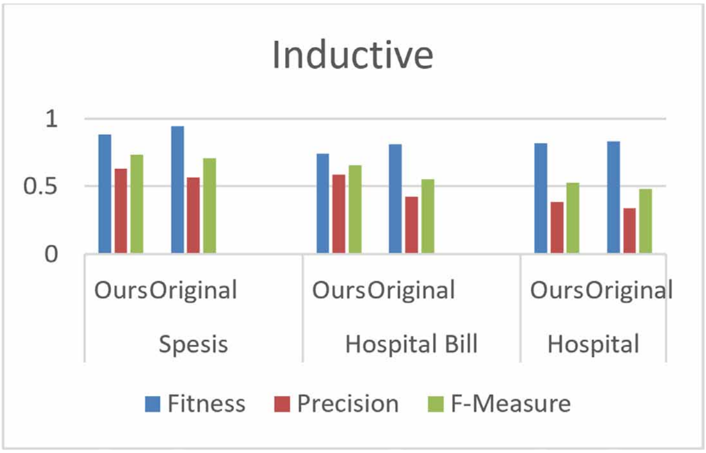
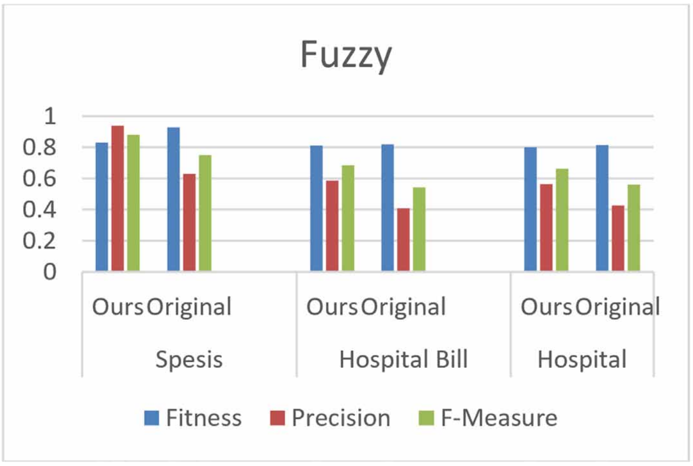


Figure 11. Real-life log comparison of the fuzzy miner algorithm



hospital billing log and hospital log, but there was a slight decrease in the F-score for the sepsis log. In Figures 10 and 11, the F-scores are improved in all three logs. These experiments indicate that the proposed filtering technique is beneficial to process discovery algorithms to obtain higher F-score values.

5. CONCLUSION

In this paper, we presented a technique to detect and remove infrequent behavior from real-life process execution logs. The core idea is to use the expectation of the event labels as a proxy for infrequent behavior.

We demonstrated the effectiveness and efficiency of our method using real-life logs and mainstream process discovery algorithms. The results indicate that the proposed approach is able to help the process discovery algorithms discover the models. Furthermore, these experiments show that our filtering method outperforms the related state-of-the-art process mining filtering techniques.

In the future, we plan to employ larger datasets to verify the effectiveness of our method and consider formal methods to validate our log filtering technique.

REFERENCES

Aggarwal, C. C. (2015). Outlier analysis. In *Data mining* (pp. 237–263). Cham: Springer.

Basu, S., & Meckesheimer, M. (2007). Automatic outlier detection for time series: An application to sensor data. *Knowledge and Information Systems*, *11*(2), 137–154. doi:10.1007/s10115-006-0026-6

Budalakoti, S., Srivastava, A. N., & Otey, M. E. (2009). Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety. *IEEE Transactions on Systems, Man and Cybernetics. Part C, Applications and Reviews*, *39*(1), 101–113. doi:10.1109/TSMCC.2008.2007248

Buijs, J. C., Van Dongen, B. F., & van Der Aalst, W. M. (2012, September). On the role of fitness, precision, generalization and simplicity in process discovery. *Proceedings of the OTM Confederated International Conferences on the Move to Meaningful Internet Systems* (pp. 305-322). Springer.

Chandola, V., Banerjee, A., & Kumar, V. (2012). Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *24*(5), 823–839. doi:10.1109/TKDE.2010.235

Conforti, R., La Rosa, M., & ter Hofstede, A. H. (2017). Filtering out infrequent behavior from business process event logs. *IEEE Transactions on Knowledge and Data Engineering*, *29*(2), 300–314. doi:10.1109/TKDE.2016.2614680

Florez-Larrahondo, G., Bridges, S. M., & Vaughn, R. (2005, September). Efficient modeling of discrete events for anomaly detection using hidden Markov models. *Proceedings of the International Conference on Information Security* (pp. 506-514). Springer. doi:10.1007/11556992_38

Günther, C. W., & Van Der Aalst, W. M. (2007, September). Fuzzy mining–adaptive process simplification based on multi-perspective metrics. *Proceedings of the International conference on business process management* (pp. 328-343). Springer.

Gupta, M., Aggarwal, C. C., & Han, J. (2011, August). Finding top-k shortest path distance changes in an evolutionary network. *Proceedings of the International Symposium on Spatial and Temporal Databases* (pp. 130-148). Springer. doi:10.1007/978-3-642-22922-0_9

Gupta, M., Gao, J., Aggarwal, C. C., & Han, J. (2014). Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *26*(9), 2250–2267. doi:10.1109/TKDE.2013.184

Gupta, M., Mallya, A., Roy, S., Cho, J. H., & Han, J. (2014, April). Local learning for mining outlier subgraphs from network datasets. *Proceedings of the 2014 SIAM International Conference on Data Mining* (pp. 73-81). Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973440.9

Gwadera, R., Atallah, M. J., & Szpankowski, W. (2005). Reliable detection of episodes in event sequences. *Knowledge and Information Systems*, *7*(4), 415–437. doi:10.1007/s10115-004-0174-5

Hofmeyr, S. A., Forrest, S., & Somayaji, A. (1998). Intrusion detection using sequences of system calls. *Journal of Computer Security*, *6*(3), 151–180. doi:10.3233/JCS-980109

Huang, Y., Li, W., Liang, Z., Xue, Y., & Wang, X. (2018). Efficient business process consolidation: Combining topic features with structure matching. *Soft Computing*, *22*(2), 645–657. doi:10.1007/s00500-016-2364-y

Keogh, E., Lonardi, S., & Chiu, B. Y. C. (2002, July). Finding surprising patterns in a time series database in linear time and space. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 550-556). ACM. doi:10.1145/775047.775128

Lane, T., & Brodley, C. E. (1997, July). Sequence matching and learning in anomaly detection for computer security. *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management* (pp. 43-49). AAAI Press.

Lane, T., & Brodley, C. E. (1999). Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, *2*(3), 295–331. doi:10.1145/322510.322526

Leemans, S. J., Fahland, D., & van der Aalst, W. M. (2013, August). Discovering block-structured process models from event logs containing infrequent behaviour. *Proceedings of the International conference on business process management* (pp. 66-78). Cham: Springer.

Măruşter, L., Weijters, A. T., Van Der Aalst, W. M., & Van Den Bosch, A. (2006). A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. *Data Mining and Knowledge Discovery*, *13*(1), 67–87. doi:10.1007/s10618-005-0029-z

Muthukrishnan, S., Shah, R., & Vitter, J. S. (2004, June). Mining deviants in time series data streams. *Proceedings. 16th International Conference on Scientific and Statistical Database Management* (pp. 41-50). IEEE.

Sani, M. F., van Zelst, S. J., & van der Aalst, W. M. (2017, September). Improving process discovery results by filtering outliers using conditional behavioural probabilities. *Proceedings of the International Conference on Business Process Management* (pp. 216-229). Cham: Springer.

Sun, P., Chawla, S., & Arunasalam, B. (2006, April). Mining for outliers in sequential databases. *Proceedings of the 2006 SIAM International Conference on Data Mining* (pp. 94-105). Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611972764.9

Van der Aalst, W. M. (2016). *Process mining: data science in action*. Springer. doi:10.1007/978-3-662-49851-4

Van der Aalst, W. M., Reijers, H. A., Weijters, A. J., van Dongen, B. F., De Medeiros, A. A., Song, M., & Verbeek, H. M. W. (2007). Business process mining: An industrial application. *Information Systems*, *32*(5), 713–732. doi:10.1016/j.is.2006.05.003

Vanden Broucke, S. K., & De Weerdt, J. (2017). Fodina: A robust and flexible heuristic process discovery technique. *Decision Support Systems*, *100*, 109–118. doi:10.1016/j.dss.2017.04.005

Weijters, A. J. M. M., & Ribeiro, J. T. S. (2011, April). Flexible heuristics miner (FHM). *Proceedings of the 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)* (pp. 310-317). IEEE.

Yankov, D., Keogh, E., & Rebbapragada, U. (2008). Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. *Knowledge and Information Systems*, *17*(2), 241–262. doi:10.1007/s10115-008-0131-9

Zhang, X., Fan, P., & Zhu, Z. (2003, August). A new anomaly detection method based on hierarchical HMM. *Proceedings of the Fourth International Conference on* Parallel and Distributed Computing, Applications and Technologies PDCAT'2003 (pp. 249-252). IEEE.

ENDNOTE

1 https://data.4tu.nl/repository/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741

Ying Huang got her PhD degree in computer software and theory from Wuhan University in 2016. She received the BS in computer application technology from Jiangxi University of Science and Technology in 2008. Currently, she works at the Gannan Normal University as a lecturer. She has published over 20 research papers in the international conferences and journals. Her research interests include the areas of service computing and business process.

Liyun Zhong got his master's degree in computer application technology from Southwest China Normal University in 2004. He received the BS in mathematics education from Gannan Normal College in 1997. Currently, he works at the Gannan Normal University as a lecturer. He has published over 10 research papers in international conferences and journals. His research interests include the areas of data mining and information management system.

Yan Chen received the Ph.D. from the college of mathematics and informatics, South China Agricultural University. Her research interests include evolutionary optimization, multi-objective optimization. She has published over 10 research papers in the international conferences and journals and has served as the program chair or program committee member at many international conferences.