Filtering out Infrequent Behavior from Business Process Event Logs

Raffaele Conforti, Marcello La Rosa and Arthur H.M. ter Hofstede

Abstract—In the era of "big data" one of the key challenges is to analyze large amounts of data collected in meaningful and scalable ways. The field of process mining is concerned with the analysis of data that is of a particular nature, namely data that results from the execution of business processes. The analysis of such data can be negatively influenced by the presence of outliers, which reflect infrequent behavior or "noise". In process discovery, where the objective is to automatically extract a process model from the data, this may result in rarely travelled pathways that clutter the process model. This paper presents an automated technique to the removal of infrequent behavior from event logs. The proposed technique is evaluated in detail and it is shown that its application in conjunction with certain existing process discovery algorithms significantly improves the quality of the discovered process models and that it scales well to large datasets.

Index Terms—Business Process Management, Process Mining, Infrequent Behavior.

1 Introduction

Process mining aims to extract actionable process knowledge from event logs of IT systems that are commonly available in contemporary organizations [31]. One area of interest in the broader field of process mining is that of *process discovery* which is concerned with the derivation of process models from event logs. Over time, a range of algorithms have been proposed that address this problem. These algorithms strike different trade-offs between the degree to which they accurately capture the behavior recorded in a log, and the complexity of the derived process model [31].

Algorithms for process discovery operate on the assumption that an event log faithfully represents the behavior of a business process as it was performed in an organization during a particular period. Unfortunately, real-life process event logs, as other kinds of events logs, often contain *outliers*. These outliers represent infrequent behavior, often referred to as "noise" [29], [30], and their presence may be exacerbated by data quality issues (e.g. data entry errors or missing data). The presence of noise leads to the derived model exhibiting execution paths that are infrequent, which clutter the model, or to models that are simply not a true representation of actual behavior. In order to limit these negative effects, process event logs are typically subjected to a pre-processing phase where they are *manually* cleaned from noise [31]. This however is a challenging and time consuming task, with no guarantee on the effectiveness of the result, especially in the context of large logs exhibiting complex process behavior [28].

The inability to effectively detect and filter out infrequent behavior has a negative effect on the quality of the discovered model, in particular on its *precision*, which is a measure of the degree to which the model allows behavior that has not been observed in the log, and its *complexity*. In fact, tests reported in this paper show that low levels of infrequent behavior already have a detrimental effect on the quality of the models produced by various discovery algorithms such as Heuristics Miner [38], Fodina [36], and Inductive Miner [23], despite these algorithms claiming to have noise-tolerant capabilities. For example, the Heuristics Miner, which employs a technique for disambiguating event dependencies, can have a 49% drop in precision when the amount of infrequent behavior corresponds to just 2% of the total log size.

This paper deals with the challenge of discovering process models of high quality in the presence of noise in event logs, by contributing an automated technique for systematically filtering out infrequent behavior from such logs. Our filtering technique first builds an abstraction of the process behavior recorded in the log as an automaton (a directed graph). This automaton captures the *direct follows* dependencies between event labels in the log. From this automaton, infrequent transitions are subsequently removed. Then the original log is replayed on this reduced automaton in order to identify events that no longer fit. These events are removed from the log. The technique aims at removing the maximum number of infrequent transitions in the automaton, while minimizing the number of events that are removed from the log. This results in a filtered log that fits the automaton perfectly.

The literature in the area of infrequent event log filtering is very scarce, offering simplistic techniques or approaches that require the availability of a reference process model as input to the filtering. To the best of our knowledge, this paper proposes the first effective technique for filtering out noise from process event logs. The novelty of the technique rests upon the choice of modeling the *infrequent log filtering problem* as an automaton. This approach enables the detection of infrequent process behavior at a fine-grain level, which leads to the removal of individual events rather than entire traces (i.e. sequences of events) from the log, hence reducing the impact on the overall process behavior captured in the log.

The technique has been implemented on top of the ProM Framework and extensively evaluated in combination with different baseline discovery algorithms, using a three-pronged approach.

- *R. Conforti, M. La Rosa and A.H.M. ter Hofstede are with the Queensland University of Technology, Australia*
  Email: {raffaele.conforti, m.larosa, a.terhofstede}@qut.edu.au
- *A.H.M. ter Hofstede is with the Eindhoven University of Technology, The Netherlands.*

First, we measured the accuracy of our technique in identifying infrequent behavior at varying levels of noise, that we injected in artificial logs. Second, we evaluated the improvement of discovery accuracy and reduction of process model complexity in the presence of varying levels of noise, for a number of baseline process discovery algorithms and compared the results with those obtained by two baseline automated filtering techniques. Third, we repeated this latter experiment using a variety of real-life logs exhibiting different characteristics such as overall size and number of (distinct) events. Discovery accuracy was measured in terms of the well-established measures of fitness and precision, while different structural complexity measures such as size, density and control-flow complexity, were used as proxies for model complexity. The results show that the use of the proposed technique leads to a statistically significant improvement of fitness, precision and complexity, while the generalization of the discovered model is not negatively affected.

As an example, Figure 1 shows two process models in the Business Process Model and Notation (BPMN) language [26], discovered from the log of a Dutch Financial Institution (BPI Challenge 2012).[1] The top model is discovered with the Inductive Miner, the bottom one is obtained by first pre-processing the log with our filtering technique, and then using the Inductive Miner. In both models, the process follows a similar execution flow: a loan application is first submitted, after which it is assessed, resulting in an acceptance or rejection. If the application is accepted, an offer is made to the customer. Despite the underpinning business process is the same, in the top model several tasks can be skipped (e.g. "Application Declined" and "Offer Created"). This results in the second model being simpler (Size = 52 nodes vs. 65) and yet more accurate (F-score = 0.671 vs. 0.551).

Finally, time performance measurements show that our technique scales well to large and complex logs, generally being able to filter a log in a few seconds.

The paper is structured as follows. Section 2 discusses algorithms for automated process discovery with a focus on their noise tolerance capabilities. Section 3 defines the proposed technique while Section 4 examines the inherent complexity of determining the minimum log automaton and proposes an Integer Linear Programming formulation to solve this problem. Section 5 is devoted to finding an appropriate threshold for determining what is to be considered infrequent behavior. Section 6 evaluates the proposed noise filtering technique, while Section 7 concludes the paper and discusses future work.

2 Background and Related Work

In this section we summarize the literature in the area of automated process model discovery, with a focus on noise-tolerance, and discuss the available metrics to measure the quality of the discovered model.

2.1 Process Log Filtering

Preprocessing a log before starting any type of process mining exercise is a de-facto practice. Typically, preprocessing includes a log filtering phase. The ProM Framework[3] offers several plugins for log filtering. In particular, two plugins deal with the removal of infrequent behavior. The *Filter Log using Simple Heuristics (SLF)* plugin removes traces which do not start and/or end with a specific event, as well as events that refer to a specific process task, on the basis of their frequency, e.g. removing all traces that start with an infrequent event.

The *Filter Log using Prefix-Closed Language (PCL)* plugin removes events from traces to obtain a log that can be expressed via a Prefix-Closed Language.[4] Specifically, this plugin keeps a trace only if it is the prefix of another trace in the log, on the basis of a user-defined frequency threshold.

Other log filtering plugins are available in ProM, which however do not specifically deal with the removal of infrequent behavior. For example, the *Filter Log by Attribute* plugin removes all events that do not have a certain attribute, or where the attribute value does not match a given value set by the user, while the *Filter on Timeframe* plugin extracts a sublog including only the events related to a given timeframe (e.g. the first six months of recordings).

In the literature, noise filtering of process event logs is only addressed by Wang et al. [37]. The authors propose an approach which uses a reference process model to repair a log whose events are affected by inconsistent labels, i.e. labels that do not match the expected behavior of the reference model. However, this approach requires the availability of a reference model.

2.2 Noise-tolerant Discovery Algorithms

The α *algorithm* [33] was the first automated process model discovery algorithm to be proposed. This algorithm is based on the *direct follows dependency* defined as $a > b$, where a and b are two process tasks and there exists an event of a directly preceding an event of b. This dependency is used to discover if one of the following relations exists between two tasks: i) *causality*, represented as $\rightarrow$, which is discovered if $a > b$ and $b \not> a$, ii) *concurrency*, represented as $\|$, which is discovered if $a > b$ and $b > a$, and iii) *conflict*, represented as #, which is discovered if $a \not> b$ and $b \not> a$. The α algorithm assumes that a log is complete and free from noise, and may produce unsound models if this is not the case.

In order to overcome the limitations of the α algorithm, including its inability to deal with noise, several noise-tolerant discovery algorithms were proposed. The first attempt was the *Heuristics Miner* [38]. This algorithm discovers a model using the α relationships. In order to limit the effects of noise, the Heuristics Miner introduces a frequency-based metric $\Rightarrow$: given two labels a and b, $a \Rightarrow b = \left(\frac{|a>b|-|b>a|}{|a>b|+|b>a|+1}\right)$. This metric is used to verify if a $\|$ relationship was correctly identified. In case the value of $\Rightarrow$ is above a given threshold, the $\|$ relationship between the two tasks is replaced by the $\rightarrow$ relationship. A similar approach is also used by *Fodina* [36], a technique which is based on the Heuristics Miner.

The *Inductive Miner* [23] is a discovery algorithm based on a divide-and-conquer approach which always results in sound process models. This algorithm, using the direct follows dependency, generates the *directly-follows graph*. Next, it identifies a cut corresponding to a particular control-flow dependency – choice ($\times$), sequence ($\rightarrow$), parallelism ($\wedge$), or iteration ($\circlearrowleft$) – in the graph

1. doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

2. Different tasks between the two models are due to the events filtering of InductiveMiner.

3. http://www.processmining.org/

4. A language is *prefix-closed* if the language is equal to the prefix closure of the language itself. For example, given a language $L = \{abc\}$, the prefix closure of L is defined as $Pref(L) = \{\varepsilon, a, ab, abc\}$

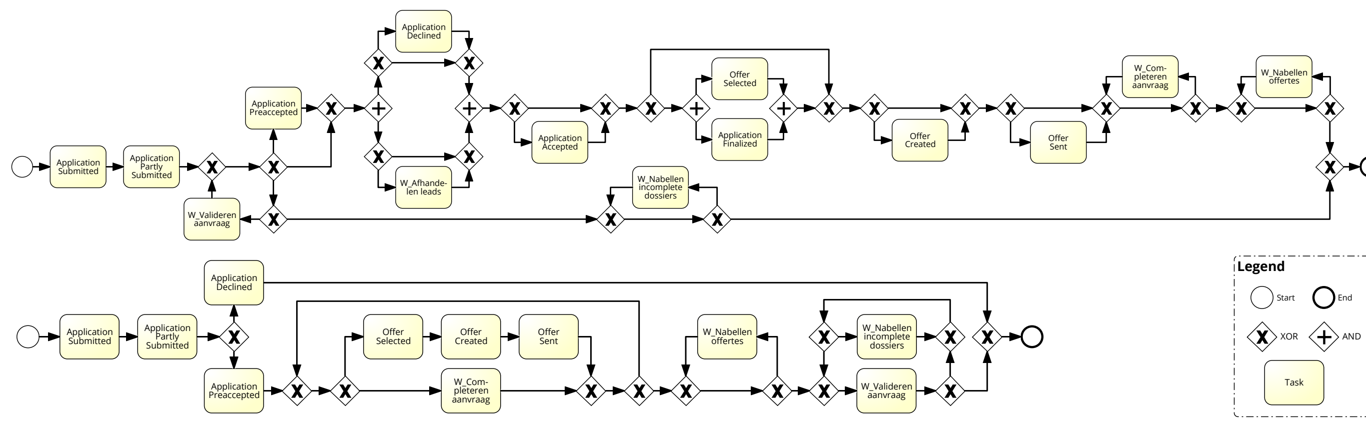


Fig. 1: BPMN model obtained using InductiveMiner on the log of a personal loan or overdraft process from a financial institution before (above) and after (below) applying the proposed technique.[2]

along which the log is split. This operation is repeated recursively until no more cuts can be identified. The mining is then performed on the portions of the log discovered using the cuts. In order to deal with noise, the algorithm applies two types of filters. The first filter behaves similarly to the Heuristics Miner and removes edges from the directly-follows graph. The second filter uses the *eventually-follows graph* to remove additional edges which the first filter did not remove.

These approaches for handling noise exhibit two limitations. First, dependencies are removed only if they are "ambiguous", e.g. replacing a $\|$ dependency with a $\rightarrow$ dependency, does not remove dependencies which are simply infrequent. Second, dependencies removed as part of the filtering stage are only removed from the dependency graph and not from the log. This influences the final result of the discovery since the algorithm may not be able to discover additional/different relationships between tasks.

The *Fuzzy Miner* [11], another discovery algorithm, applies noise filtering a posteriori, directly on the model discovered. This algorithm is based on the concepts of correlation and significance, and produces a *fuzzy net* where each node and edge is associated with a value of correlation and significance. After the mining phase, one can provide a significance threshold and a correlation threshold which are used for filtering. These two thresholds can simplify the model by preserving highly significant behavior, aggregating less significant but highly correlated behavior (via clustering of nodes and edges), and abstracting less significant and less correlated behavior (via removal of nodes and edges). The main problem of this algorithm is that a fuzzy net only provides an abstract representation of the process behavior extracted from the log, due to its intentionally underspecified semantics, which leaves room for interpretation.

Finally, the *ILP miner* [35] follows a different approach in order to handle noise. In this case noise is not filtered out but is integrated in the discovered model. This algorithm translates relations observed in the logs into an Integer Linear Programming (ILP) problem, where the solution is a Petri net capable of reproducing all behavior present in the log (noise included). The negative effect of this approach is that it tends to generate "flower-like"[5] models which suffer from very low precision.

5. This terms derives from the resemblance of the model with a flower, where a place in the center is surrounded by several transitions which with their arcs resemble the shapes of petals.

2.3 Outlier Detection in Data Mining

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) describing the normal behavior, and consider as outliers all data points which deviate from this model [4].

In the context of temporal data, these algorithms have been extensively surveyed by Gupta et al. [13] (for events with continuous values, a.k.a. *time series*) and by Chandola et al. [7] (for events with discrete values, a.k.a. *discrete sequences*).

According to [13], we can classify those approaches into three major groups. The first group encompasses approaches dealing with the problem of detecting if an entire sequence of events is an outlier. These approaches either build a model from the entire dataset, i.e. from all sequences (e.g. [6], [10], [27], [40]), or subdivide the dataset into overlapping windows and build a model for each window (e.g. [16], [21], [22]). While approaches of this type can in principle be used for filtering out infrequent process behavior in event logs, their filtering would be too coarse-grained, as it would lead to the removal of entire traces in the log, impacting as a result, on the accuracy of the discovered process model.

Approaches in the second group identify as outliers single data points (e.g. [5], [9], [25]) or sequences thereof (e.g. [18], [39]) 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 however would lead to mixing up 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 process events are not repeated often within a trace, their relative frequency would be very low, leading to considering almost all events of a trace as outliers.

Finally, approaches in the third group identify anomalous patterns within sequences (e.g. [15], [19]). 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 which can be used as input.

Outlier detection algorithms based on graphs, e.g. [14] and [12], share some similarities with our approach. Given a graph as input (e.g. a co-authorship graph), which could be built from the log, they identify outlier subgraphs within that graph. These approaches consider undirected graphs where the order dependency between elements is irrelevant. For example, if the subgraph made of events “A” and “B” is considered as an outlier, these two events will be removed from each trace of the log in which they occur, regardless of their relative order or frequency. While this filtering mechanism may work with process event logs, the removal of infrequent behavior would again be too coarse-grained.

2.4 Model Dimensions

The quality of a discovered model can be measured according to four dimensions: fitness (recall), precision (appropriateness), generalization, and complexity.

Fitness measures how well a model can reproduce the process behavior contained in a log. A fitness measurement of 0 indicates the inability to reproduce any of the behavior recorded in the log while a value of 1 indicates the ability to reproduce all recorded behavior. In order to measure fitness we use the approach proposed by Adriansyah et al. [3] which, after aligning a log to a model, measures the number of times the two are not moving synchronously. This approach is widely accepted as the main fitness measurement [8], [23].

Precision measures the degree to which the behavior made possible by a model is found in a log. A value of 0 indicates that the model can produce a lot of behavior not observed in the log while a value of 1 indicates that the model only allows behavior observed in the log. In order to obtain a measurement of precision that is consistent with that of fitness, we decided to adopt the approach proposed by Adriansyah et al. [2]. Accordingly, after generating an alignment automaton describing the set of executed actions and the set of possible actions, this approach measures precision based on the ratio between the number of executed actions over the number of possible actions.

The F-score is often used to combine fitness and precision in a single measure of model *accuracy*, and is the harmonic mean of fitness and precision $\left(2 \cdot \frac{Fitness \cdot Precision}{Fitness + Precision}\right)$.

Generalization can be seen as the opposite of precision. It provides a measure of the capability of a model to produce behavior not observed in the log. We decided to measure generalization using 10-fold cross validation, which is an established approach in data mining [20]. Accordingly, a log is divided into ten parts and each part is used to measure the fitness of the model generated using the remaining nine parts. Another approach for measuring generalization is the approach proposed by van der Aalst et al. [32] which we decided not to use since in our tests this approach returns similar results across all discovered models.

Finally, *complexity* quantifies the structural complexity of a process model and can be measured using various complexity metrics [24] such as:

- *Size:* the number of nodes.
- *Control-Flow Complexity (CFC):* the amount of branching caused by gateways in the model.
- *Average Connector Degree (ACD):* the average number of nodes a connector is connected to.
- *Coefficient of Network Connectivity (CNC):* the ratio between arcs and nodes.
- *Density:* the ratio between the actual number of arcs and the maximum possible number of arcs in a model.

Noise affects the above quality dimensions in different ways. Recall is not reliable when computed on a log containing noise as behavior made possible by the discovered model is not necessarily behavior that reflects reality. The presence of noise tends to lower precision as this noise introduces spurious connections between event labels. Given the dual nature of precision and generalization it is clear that the presence of noise increases generalization, as the new connections lead to a model allowing more behavior. Finally, the complexity of discovered process models for logs with noise tends to be higher due to the increased number of tasks and arcs resulting from the presence of new connections introduced by the noise.

Symbol	Meaning
Γ	Finite set of Tasks
$\mathscr{E}$	Finite set of Events
$\mathscr{C}$	Finite set of Trace Identifiers
C	Surjective function linking $\mathscr{E}$ to $\mathscr{C}$
T	Surjective function linking $\mathscr{E}$ to Γ
$<$	Strict total ordering over Events
$\sqsubset$	Strictly Before Relation
$\rightsquigarrow$	Direct Follow Dependency
$\mathscr{A}$	Log Automaton
$\#_\Gamma$	Function counting occurencies of Γ (States)
$\#_\rightsquigarrow$	Function counting occurencies of $\rightsquigarrow$ (Arcs)
$\rightsquigarrow^m$	Set of Frequent arcs
$\rightsquigarrow^o$	Set of Infrequent arcs
ε	Frequency Threshold
Γ^R	Set of required States
$\uparrow_{\mathscr{A}}$	Set of initial States
$\downarrow_{\mathscr{A}}$	Set of final States
Φ	Possible Arc Sets
$\rightharpoondown$	Minimal Arc Set
$\mathscr{A}^f$	Anomaly-Free Automaton
$\hookrightarrow$	Replay of two Events
replayable	Replayable sequence of Events
Θ^c	Subtrace
$\mathscr{F}$	Filtered Log

TABLE 1: Formal notation.

3 Approach

In this section we present our technique for the filtering of infrequent behavior. After introducing preliminary concepts such as event log and direct follow dependencies, the concept of log automaton is presented. The identification of infrequent behavior in a log automaton and its removal conclude the section.

The formal notation used in this section is summarized in Table 1.

3.1 Preliminaries

For auditing and analysis purposes, the execution of business processes supported by IT systems is generally recorded in system or application logs. These logs can then be converted into event logs for process mining analysis. An *event log* is composed of a set of *traces*. Each trace (a.k.a. *case*) captures the footprint of a process instance in the log, in the form of a sequence of events, and is identified by a unique case identifier. Each *event* records the execution of a specific process task within a trace. For instance, with reference to the log of the personal loan and overdraft process of Figure 1, there will be events recording the execution of task “Application Submitted” and “Application Partly Submitted”. These are the first two events of each trace in this log, since the corresponding tasks are the first two tasks to be always executed as part of this business process, as shown in the two models of Figure 1. For this process, the case identifier is the loan or overdraft application number, which is unique for each trace.

Definition 1 (Event Log)**.** *Let Γ be a finite set of tasks. A log $\mathscr{L}$ is defined as $\mathscr{L} = (\mathscr{E}, \mathscr{C}, C, T, <)$ where $\mathscr{E}$ is the set of events, $\mathscr{C}$ is the set of case identifiers, $C : \mathscr{E} \to \mathscr{C}$ is a surjective function linking events to cases, $T : \mathscr{E} \to \Gamma$ is a surjective function linking events to tasks, and $< \subseteq \mathscr{E} \times \mathscr{E}$ is a strict total ordering over the events.*

The *strictly before* relation $\sqsubset$ is a derived relation over events, where $e_1 \sqsubset e_2$ holds iff $e_1 < e_2 \wedge C(e_1) = C(e_2) \wedge \nexists e_3 \in \mathscr{E}[C(e_3) = C(e_1) \wedge e_1 < e_3 \wedge e_3 < e_2]$.

Given a log, several relations between tasks can be defined based on their underlying events. We are interested in the *direct follow dependency*, which captures whether a task can directly follow another task in the log.

Definition 2 (Direct Follow Dependency)**.** *Given tasks $x, y \in \Gamma$, x* directly follows *y, i.e. $x \rightsquigarrow y$, iff $\exists e_1, e_2 \in \mathscr{E} \wedge T(e_1) = x \wedge T(e_2) = y \wedge e_1 \sqsubset e_2$.*

3.2 Infrequent behavior detection

In this section we present a technique for infrequent behavior detection which relies on the identification of anomalies in a so-called *log automaton*. In this context, anomalies represent relations, which occur infrequently.

An automaton is a directed graph where each node (here referred to as a state) represents a task which can occur in the log under consideration and each arc connecting two states indicates the existence of a direct follow dependency between the corresponding tasks.

Definition 3 (Log Automaton)**.** *A log automaton for an event log $\mathscr{L}$ is defined as a directed graph $\mathscr{A} = (\Gamma, \rightsquigarrow)$.*

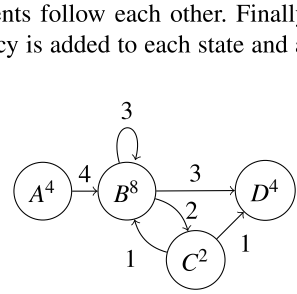
For an automaton we can retrieve all initial states through $\uparrow_{\mathscr{A}} = \{x \in \Gamma \mid \nexists y \in \Gamma[y \rightsquigarrow x]\}$ and all final states through $\downarrow_{\mathscr{A}} = \{x \in \Gamma \mid \nexists y \in \Gamma[x \rightsquigarrow y]\}$.

As we are interested in frequencies of task occurrences and of direct follow dependencies, we introduce the function $\#_\Gamma : \Gamma \to \mathbb{N}$ defined by $\#_\Gamma(x) = |\{z \in \mathscr{E} \mid T(z) = x\}|$ and the function $\#_\rightsquigarrow : \rightsquigarrow \to \mathbb{N}$ defined by $\#_\rightsquigarrow(x, y) = |\{(e_1, e_2) \in \mathscr{E} \times \mathscr{E} \mid T(e_1) = x \wedge T(e_2) = y \wedge e_1 \sqsubset e_2\}|$.

Figure 2 shows how to generate a log automaton from a log. Each event in the log is converted into a state, with A as initial state and D as final state. Moreover, two states are connected with an arc if in the log the two events follow each other. Finally, an annotation showing the frequency is added to each state and arc.

Log
$<A, B, B, D>$
$<A, B, C, D>$
$<A, B, B, B, D>$
$<A, B, C, B, D>$

$\Rightarrow$



Log Automaton

Fig. 2: Example: Log Automaton.

An arc is considered *infrequent* iff its *relative frequency* is a value smaller than a given threshold ε where the relative frequency of an arc is computed by dividing the frequency of the arc by the sum of the frequencies of the source and target states.

Definition 4 (Infrequent and Frequent Arcs)**.** *The set of infrequent arcs $\rightsquigarrow^o$ is defined as $\{(x,y) \in \Gamma \times \Gamma \mid (2 \cdot \#_\rightsquigarrow(x,y)/(\#_\Gamma(x) + \#_\Gamma(y)) < \varepsilon) \wedge x \rightsquigarrow y\}$. The complement of this set is the set of* frequent arcs *defined by $\rightsquigarrow^m \triangleq \rightsquigarrow \setminus \rightsquigarrow^o$.*

Considering the example in Figure 2 and using a threshold of 0.3, the set of infrequent arcs contains the arcs (C, B), with a relative frequency of $\frac{2 \cdot 1}{2+8} = 0.2 < 0.3$, and (C, D), with a relative frequency of $\frac{2 \cdot 1}{2+4} = 0.25 < 0.3$.

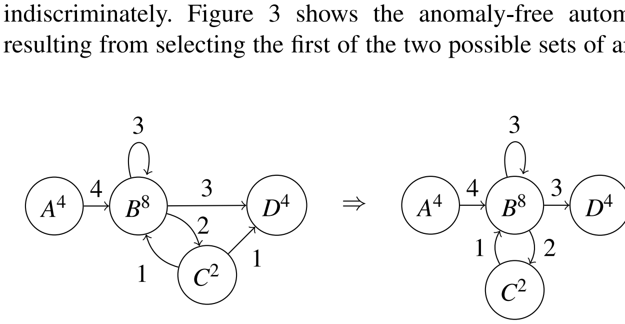
The indiscriminate removal of anomalies from a log automaton may result in a log automaton where certain states can no longer be reached from an initial state or from which final states can no longer be reached. This loss of connectivity may in some instances be considered acceptable but not in others as there may be states that should be retained from a stakeholder's perspective.

Definition 5 (Required States)**.** *Given a log automaton $\mathscr{A}$, the states that need to be preserved during reduction (i.e. the process of removing infrequent transitions) are referred to as the* required states *and the corresponding set of these states is denoted as Γ^R and thus $\Gamma^R \subseteq \Gamma$. These states need to be identified by a stakeholder, but must include all initial and all final states, i.e. $\uparrow_{\mathscr{A}} \subset \Gamma^R$ and $\downarrow_{\mathscr{A}} \subset \Gamma^R$.*

From now on, we consider a log automaton as $\mathscr{A} = (\Gamma, \Gamma^R, \rightsquigarrow^m, \rightsquigarrow^o)$.

In order to obtain an anomaly-free automaton $\mathscr{A}^f$ where the connectivity of required states is not lost, we first consider the set Φ which consists of *possible arc sets* and which is defined by $\Phi \triangleq \{\rightharpoondown \in \mathscr{P}(\rightsquigarrow) \mid \rightsquigarrow^m \subset \rightharpoondown \wedge \forall s \in \Gamma^R\ \exists a \in \uparrow_{(\Gamma, \rightharpoondown)} [a \rightharpoondown^+ s] \wedge \forall s \in \Gamma^R\ \exists a \in \downarrow_{(\Gamma, \rightharpoondown)} [s \rightharpoondown^+ a]\}$.[6] We are interested in a (there are potentially multiple candidates) minimal set $\rightharpoonup$ in Φ, i.e. a set from which no more infrequent arcs can be removed. Hence, $\rightharpoonup \in \Phi$ and for all $E \in \Phi : |E| \geq |\rightharpoonup|$. The set $\rightharpoonup$ is then used to generate our anomaly-free automaton $\mathscr{A}^f \triangleq (\Gamma, \Gamma^R, \rightsquigarrow^m, \rightsquigarrow^o \cap \rightharpoonup)$.

In the example provided in Figure 2, assuming that all states are required and using a threshold ε of 0.3, the set Φ contains two sets of arcs. The first set contains the arcs (A, B), (B, C), (B, D) and (C, B), while the second set contains the arcs connecting (A, B), (B, C), (B, D) and (C, D). In this case since the size of the two sets is the same we can choose one of the two sets indiscriminately. Figure 3 shows the anomaly-free automaton resulting from selecting the first of the two possible sets of arcs.



Log Automaton *Anomaly-Free Automaton*

Fig. 3: Example: Anomaly-Free Automaton.

6. $\rightharpoondown^+$ is the transitive closure of $\rightharpoondown$.

3.3 Infrequent behavior removal

In this section focus is on the removal of infrequent behavior from a log using an automaton from which anomalies have been removed as described in the previous section.

The idea behind our technique is inspired by the observation that infrequent behavior in an event log is often caused by events that are recorded in the wrong order or at an incorrect point in time. Such infrequent behavior may cause the derivation of direct follow dependencies that in fact do not hold or may cause direct follow dependencies that hold to be overlooked. Hence, our starting point for the removal of infrequent behavior is to focus on incorrectly recorded events. To this end, events that cannot be replayed on the anomaly-free automaton are removed.

Definition 6 (Replayable)**.** *Given a set of events $E \subseteq \mathscr{E}$ and an anomaly-free automaton $\mathscr{A}^f$, this automaton can* replay *a sequence of two events $e, e' \in E$, i.e. $e \hookrightarrow_E e'$, iff $\exists x, y \in \Gamma[x = T(e) \wedge y = T(e') \wedge x \rightsquigarrow y]$. The automaton can replay a set of events E, i.e. replayable(E), iff there is a sequence $e_1, e_2, \ldots, e_n$ with $E = \{e_1, e_2, \ldots, e_n\}$, and $e_1 \hookrightarrow_E e_2$, $e_2 \hookrightarrow_E e_3$, $\ldots$, $e_{n-1} \hookrightarrow_E e_n$, and e_1 is an event corresponding to an initial state, $T(e_1) \in \uparrow_{\mathscr{A}^f}$, and e_n is an event corresponding to a final state, $T(e_n) \in \downarrow_{\mathscr{A}^f}$.*

Having defined what it means to be able to replay a trace, we can identify the subtraces of a trace that can be replayed.

Definition 7 (Subtrace)**.** *Given a trace corresponding to case c, the set of its* subtraces *Θ^c is defined as $\Theta^c \triangleq \{E \in \mathscr{P}(\mathscr{E}^c) \mid$ replayable$(E)\}$, where $\mathscr{E}^c$ is the set of events in case c, i.e. $\{e \in \mathscr{E} \mid C(e) = c\}$.*

Among the set of replayable subtraces we are interested in the ones that are the longest.

Definition 8 (Longest Replayable Subtrace)**.** *Given a trace corresponding to case c, a set of its* longest replayable subtraces *θ^c is defined as $\theta^c \in \Theta^c$ such that for all $\eta \in \Theta^c$ it holds that $|\theta^c| \geq |\eta|$.*

Given an anomaly-free automaton $\mathscr{A}^f$, the filtered log $\mathscr{F}$ is defined as the set of the longest subtraces of $\mathscr{L}$ which can be replayed by $\mathscr{A}^f$.

Definition 9 (Filtered Log)**.** *The* filtered version *of log $\mathscr{L}$ is defined as $\mathscr{F} = (E, ran(C_{\restriction_E}), C_{\restriction_E}, T_{\restriction_E}, < \cap\, E \times E)$ where E is defined as $\bigcup_{c \in \mathscr{C}} \theta^c$.*

Figure 4 shows how to use an anomaly-free automaton to generate a filtered log. Starting from a log containing infrequent behavior, the log automaton is generated, where A is the initial state and D the final state. Using a threshold ε of 0.3, the anomaly-free automaton is derived, and then used to filter the log. In the filtered log, event C is removed from the second trace since the anomaly-free automaton could not reproduce this event while reaching the final state, which was thus treated as infrequent behavior.

4 The Minimum Anomaly-Free Automaton Problem

In this section first we prove the inherent complexity of determining a minimum anomaly-free automaton and then we provide an ILP formulation of the problem.

4.1 Complexity

The identification of the minimum anomaly-free automaton is an NP-hard problem. In this section we provide a proof of its complexity presenting a polynomial time transformation from the set covering problem (a well known NP-complete problem [17]) to the minimum anomaly-free automaton problem.

The set covering problem is the problem of identifying the minimum number of sets required to contain all elements of a given universe. Formally, given a universe U and a set $S \subseteq \mathscr{P}(U)$ composed of subsets of U, an instance (U, S) of the set covering problem consists of identifying the smallest subset of S, $C \subseteq S$, such that its union equals U, i.e. $\bigcup C = U$.

Formally, let $I = (U, S)$ be an instance of a set cover problem, its related anomaly-free automaton problem is defined as $\Pi(I) = (\Gamma_I, \Gamma_I^R, \rightsquigarrow_I^m, \rightsquigarrow_I^o)$ where:

- $\Gamma_I \triangleq \Gamma_S \cup \Gamma_U \cup \{i, o\}$ with
  - $\Gamma_S \triangleq \{s_j \mid j \in S\}$;
  - $\Gamma_U \triangleq \{u_k \mid k \in U\}$;
- $\Gamma_I^R \triangleq \Gamma_U \cup \{i, o\}$;
- $\rightsquigarrow_I^m \triangleq \{(s_j, u_k) \mid j \in S \wedge k \in j\} \cup \Gamma_U \times \{o\}$;
- $\rightsquigarrow_I^o \triangleq \{i\} \times \Gamma_S$.

This construction is only applicable when the set covering problem has a solution. Checking if a set covering problem has a polynomial time complexity and can be checked by verifying if the union of all sets in S is equal to U, i.e. $\bigcup S = U$.

Proposition 1. *If a set covering problem $I = (U, S)$ has a solution (this can be checked in polynomial time) then $\Pi(I)$ is an anomaly-free automaton.*

Lemma 1. *Let $I = (U, S)$ be an instance of the set covering problem that has a solution and let $\Pi(I) \triangleq (\Gamma_I, \Gamma_I^R, \rightsquigarrow_I^m, \rightsquigarrow_I^o)$ be its transformation. If $\mathscr{A}_I' = (\Gamma_I, \Gamma_I^R, \rightsquigarrow_I^m, \rightsquigarrow_I^{o'})$ is a minimum anomaly-free automaton for $\Pi(I)$ then $C \triangleq \{c \in S \mid i \rightsquigarrow_I^{o'} s_c\}$ is a minimum set cover for I.*

Proof. C is a minimum set cover for I iff C is a cover, and C is minimal.

1) C is a cover. Let $k \in U$. Consider u_k in $\mathscr{A}_I'$, u_k is on a path from i to o. Hence, there is a node s_j such that $i \rightsquigarrow_I^{o'} s_j$ and $s_j \rightsquigarrow_I^m u_k$. Hence, $j \in C$ and $k \in j$. Therefore $k \in \bigcup C$ and hence $U \subseteq \bigcup C$ and thus C is a cover.
2) C is minimal. Let us assume C is not minimal. Then there exists a cover $C' \subseteq S$ such that $|C'| < |C|$. Define $\mathscr{A}_I'' = (\Gamma_I, \Gamma_I^R, \rightsquigarrow_I^m, \rightsquigarrow_I^{o''})$ with $\rightsquigarrow_I^{o''} = \{(i, s_c) \mid c \in C'\}$. Observe that $|\rightsquigarrow_I^{o''}| < |\rightsquigarrow_I^{o'}|$. Let $k \in U$. As C' is a cover, there exists a j in C' such that $k \in j$. Therefore, $s_j \rightsquigarrow_I^m u_k$. As $i \rightsquigarrow_I^{o''} s_j$ (given that $j \in C'$) and $u_k \rightsquigarrow_I^m o$ (by construction) u_k is on a path from i to o in $\mathscr{A}_I''$. Hence, i, o, and all states u_k, $k \in U$, are on a path from i to o. Therefore in $\mathscr{A}_I''$ all required states are on a path from i to o and $\mathscr{A}_I''$ contains fewer infrequent arcs than $\mathscr{A}_I'$. Hence $\mathscr{A}_I'$ is not minimal. Contradiction.

□

Corollary 1.1. *The minimum anomaly-free automaton problem is NP-hard.*

This follows from the fact that transforming a set covering problem to a minimum anomaly-free automaton problem is a polynomial time transformation, and the fact that through this transformation we can solve a set covering problem through a search for a minimum anomaly-free automaton (see Lemma 1).

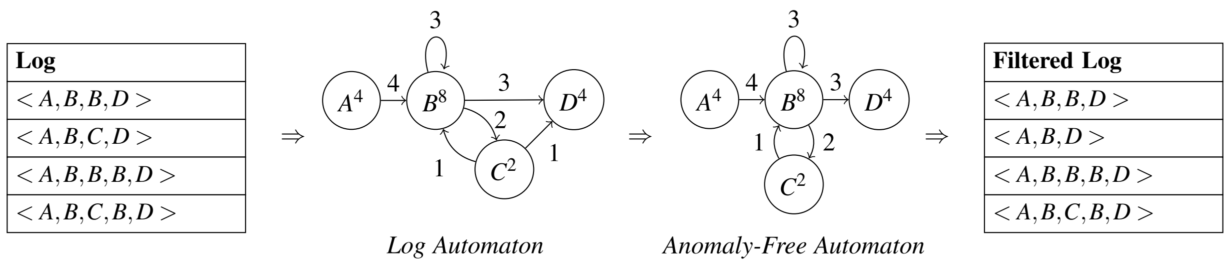


Fig. 4: Example: Anomaly-Free Automaton.

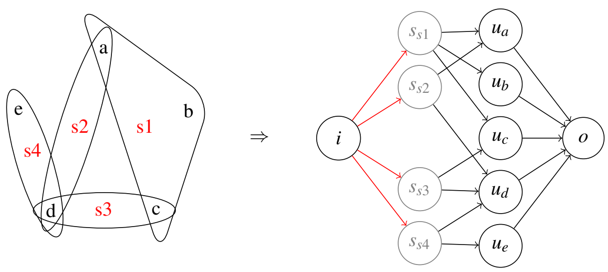


Fig. 5: Sample Reduction from Set Covering Problem to Minimum Anomaly-Free Automaton Problem.

Lemma 2. *Let $I = (U,S)$ be an instance of the set covering problem and let $\Pi(I) \triangleq (\Gamma_I, \Gamma_I^R, \rightsquigarrow_I^m, \rightsquigarrow_I^o)$ be its transformation. If $C \in S$ is a minimum set cover for I then $\mathscr{A}'_I = (\Gamma_I, \Gamma_I^R, \rightsquigarrow_I^m, \rightsquigarrow_I^{o'})$, where $\rightsquigarrow_I^{o'} = \{(i, s_c) \mid c \in C\}$, is a minimum anomaly-free automaton for $\Pi(I)$.*

Proof. $\mathscr{A}'_I$ is a minimum anomaly-free automaton for $\Pi(I)$ iff in $\mathscr{A}'_I$ all required states are on a path from source (i) to sink (o), and $\mathscr{A}'_I$ is minimal.

1) All required states are on a path from i to o in $\mathscr{A}'_I$. Let $k \in U$. As C is a cover, there exists a j in C such that $k \in j$. Therefore, $s_j \rightsquigarrow_I^m u_k$. As $i \rightsquigarrow_I^{o'} s_j$ (given that $j \in C$) and $u_k \rightsquigarrow_I^m o$ (by construction) u_k is on a path from i to o in $\mathscr{A}'_I$. Hence, i, o, and all states u_k, $k \in U$, are on a path from i to o. Therefore in $\mathscr{A}'_I$ all required states are on a path from i to o.
2) $\mathscr{A}'_I$ is minimal. Let us assume $\mathscr{A}'_I$ is not minimal and there exists a minimum anomaly-free automaton $\mathscr{A}''_I = (\Gamma_I, \Gamma_I^R, \rightsquigarrow_I^m, \rightsquigarrow_I^{o''})$ such that $|\rightsquigarrow_I^{o''}| < |\rightsquigarrow_I^{o'}|$. Define $C' = \{c \in S \mid i \rightsquigarrow_I^{o''} s_c\}$. Observe $|C'| < |C|$. Since in $\mathscr{A}''_I$ all required states are on a path from i to o, C' is a cover (see Lemma 1). Hence C is not a minimum set cover. Contradiction.

□

Figure 5 shows how to reduce a set covering problem to a minimum anomaly-free automaton problem. In the example, the universe U contains five elements, $U = \{a,b,c,d,e\}$. Among these elements four subsets are defined $s1 = \{a,b,c\}$, $s2 = \{a,d\}$, $s3 = \{c,d\}$, and $s4 = \{d,e\}$. As part of the reduction an initial state i and a final state o are introduced, and each element is converted into a state. Additionally, states in the form u_k are connected to the final state. Moreover, for each subset a state is introduced connecting it to the states representing elements of the subset, for example S_{s1} is connected to u_a, u_b, and u_c. Finally, the initial state is connected to each state representing a subset through an infrequent arc (red arrow).

4.2 ILP Formulation

Through the application of Integer Linear Programming (ILP) one can effectively determine a minimum log automaton of a given log. In the following we show how to formulate this problem as an ILP problem. Before presenting the formulation the following set of variables needs to be introduced:

- for each arc $n_1 \rightsquigarrow n_2$ there exists a variable $e_{n_1,n_2} \in \{0,1\}$. If the solution of the ILP problem is such that $e_{n_1,n_2} = 1$, the minimum automaton contains an arc connecting n_1 to n_2.
- for each state $n \in \Gamma$ there exists a variable $\overleftarrow{\mathscr{C}}_n \in \{0,1\}$. If state n is reachable from an initial state, the ILP solution assigns to $\overleftarrow{\mathscr{C}}_n$ a value of 1; otherwise $\overleftarrow{\mathscr{C}}_n = 0$.
- for each state $n \in \Gamma$ there exists a variable $\overrightarrow{\mathscr{C}}_n \in \{0,1\}$. If state n can reach a final state, the ILP solution assigns to $\overrightarrow{\mathscr{C}}_n$ a value of 1; otherwise $\overrightarrow{\mathscr{C}}_n = 0$.
- for each arc $n_1 \rightsquigarrow n_2$ there exists a variable $\overleftarrow{\mathscr{L}}_{n_2,n_1} \in \{0,1\}$. If state n_2 is reachable from an initial state through an arc connecting n_1 to n_2, the ILP solution assigns to $\overleftarrow{\mathscr{L}}_{n_2,n_1}$ a value of 1; otherwise $\overleftarrow{\mathscr{L}}_{n_2,n_1} = 0$.
- for each arc $n_1 \rightsquigarrow n_2$ there exists a variable $\overrightarrow{\mathscr{L}}_{n_1,n_2} \in \{0,1\}$. If state n_1 can reach a final state through an arc connecting n_1 to n_2, the ILP solution assigns to $\overrightarrow{\mathscr{L}}_{n_1,n_2}$ a value of 1; otherwise $\overrightarrow{\mathscr{L}}_{n_1,n_2} = 0$.

The ILP problem aims at minimizing the number of arcs of an automaton:

$$\min \sum_{n_1 \in N} \sum_{\substack{n_2 \in N \\ n_1 \rightsquigarrow n_2}} e_{n_1,n_2}. \tag{1}$$

This ILP problem is subject to the following constraints:

- For each frequent arc we impose that a solution must contain it:
$$e_{n_1,n_2} = 1. \tag{2}$$
- For each initial state $s \in \uparrow_{\mathscr{A}}$, we mark it as reachable from an initial state:
$$\overleftarrow{\mathscr{C}}_s = 1. \tag{3}$$
- For each final state $t \in \downarrow_{\mathscr{A}}$, we mark it as able to reach a final state:
$$\overrightarrow{\mathscr{C}}_t = 1. \tag{4}$$
- For each state $n \in \Gamma^R$, we require it to be reachable from an initial state:
$$\overleftarrow{\mathscr{C}}_n = 1. \tag{5}$$
- For each state $n \in \Gamma^R$, we require it to be able to reach a final state:
$$\overrightarrow{\mathscr{C}}_n = 1. \tag{6}$$

- For each couple of states $n_1, n_2 \in \Gamma$, if n_1 is reachable from an initial state and n_1 is connected to n_2, $n_1 \rightsquigarrow n_2$, then n_2 is reachable from an initial state through n_1:

$$\overleftarrow{\mathscr{L}}_{n_2,n_1} = 1 \Leftrightarrow \overleftarrow{\mathscr{C}}_{n1} + e_{n_1,n_2} = 2. \tag{7}$$

- For each couple of states $n_1, n_2 \in \Gamma$, if n_2 can reach a final state and n_1 is connected to n_2, $n_1 \rightsquigarrow n_2$, then n_1 is able to reach a final state through n_2:

$$\overrightarrow{\mathscr{L}}_{n_1,n_2} = 1 \Leftrightarrow \overrightarrow{\mathscr{C}}_{n2} + e_{n_1,n_2} = 2. \tag{8}$$

- Each non-initial state $n_1 \in \Gamma \setminus \uparrow_{\mathscr{A}}$ is reachable from an initial state if and only if there is at least one path from an initial state to that state that uses an arc originating from a state reachable from an initial state:

$$\overleftarrow{\mathscr{C}}_{n_1} = 1 \Leftrightarrow \sum_{\substack{n_2 \in N \\ n_1 \neq n_2}} \overleftarrow{\mathscr{L}}_{n_1,n_2} \geq 1. \tag{9}$$

- Each non-final state $n_1 \in \Gamma \setminus \downarrow_{\mathscr{A}}$ can reach a final state if and only if at least one of the target states of its outgoing arcs can reach a final state:

$$\overrightarrow{\mathscr{C}}_{n_1} = 1 \Leftrightarrow \sum_{\substack{n_2 \in N \\ n_1 \neq n_2}} \overrightarrow{\mathscr{L}}_{n_1,n_2} \geq 1. \tag{10}$$

In the following lemmas we show how the constraints in Equivalences 7-10 can be translated into an equivalent set of linear constraints.

Lemma 3. *Constraints of the form as in Equivalence 7 can be rewritten into a set of equivalent inequalities of the form shown below:*

$$\begin{aligned} 2 \cdot \overleftarrow{\mathscr{L}}_{n_2,n_1} - \overleftarrow{\mathscr{C}}_{n1} - e_{n_1,n_2} &\leq 0, \\ \overleftarrow{\mathscr{C}}_{n1} + e_{n_1,n_2} - 2 \cdot \overleftarrow{\mathscr{L}}_{n_2,n_1} &\leq 1. \end{aligned} \tag{11}$$

Proof. Let us introduce the following equalities for readability purposes:

$$\begin{aligned} \overleftarrow{\mathscr{L}}_{n_2,n_1} &= x, \\ \overleftarrow{\mathscr{C}}_{n_1} + e_{n_1,n_2} &= y. \end{aligned}$$

Now we can rewrite Inequalities 11 as:

$$\begin{aligned} 2 \cdot x - y &\leq 0, \\ y - 2 \cdot x &\leq 1. \end{aligned}$$

Considering that x can only be 0 or 1, and y can only be 0, 1, or 2, the first constraint can only be satisfied if either $x = 0$ or if $y = 2$. Similarly, the second constraint can only be satisfied if either $y < 2$ or if $x = 1$. We can see that in order to satisfy both constraints either $x = 0$ and $y \neq 2$ or $x = 1$ and $y = 2$ which is exactly the constraint defined in Equivalence 7. □

Lemma 4. *Constraints of the form as in Equivalence 9 can be rewritten into a set of equivalent inequalities of the form shown below:*

$$\begin{aligned} \overleftarrow{\mathscr{C}}_{n_1} - \sum_{\substack{n_2 \in N \\ n_1 \neq n_2}} \overleftarrow{\mathscr{L}}_{n_1,n_2} &\leq 0, \\ \sum_{\substack{n_2 \in N \\ n_1 \neq n_2}} \overleftarrow{\mathscr{L}}_{n_1,n_2} - M \cdot \overleftarrow{\mathscr{C}}_{n_1} &\leq 0. \end{aligned} \tag{12}$$

where M is a sufficiently large number (e.g. the largest machine-representable number).

Proof. Let us introduce the following inequalities for readability purposes:

$$\begin{aligned} \overleftarrow{\mathscr{C}}_{n_1} &= x, \\ \textstyle\sum_{\substack{n_2 \in N \\ n_1 \neq n_2}} \overleftarrow{\mathscr{L}}_{n_1,n_2} &= y. \end{aligned}$$

Now we can rewrite Inequalities 12 as:

$$\begin{aligned} x - y &\leq 0, \\ y - M \cdot x &\leq 0. \end{aligned}$$

Considering that x can only be 0 or 1, and $y \geq 0$, the first constraint can only be satisfied if either $x = 0$ or if $y \geq 1$. Similarly, the second constraint can only be satisfied if either $y = 0$ or if $x = 1$. We can see that in order to satisfy both constraints either $x = 0$ and $y = 0$ or $x = 1$ and $y \geq 1$ which is exactly the constraint defined in Equivalence 9. □

5 Frequency Threshold Identification

In section 3.2 we introduced the concepts of frequent and infrequent arcs in a log automaton. These concepts are based on a *frequency threshold* ε. In this section we present a technique for the automated identification of such a threshold.

The idea behind the technique is that in an optimal scenario arc frequencies will have a symmetrical distribution shifted toward 1, while the presence of infrequent arcs produces, in the worst case, a positively skewed distribution shifted toward 0.

In order to identify the optimal frequency threshold we need to introduce the concepts of *lower-half interquartile range* (IQR_L) and *upper-half interquartile range* (IQR^U). These two concepts are based on the *interquartile range* (IQR) defined as the difference between the upper and the lower quartiles, i.e. $IQR = Q_3 - Q_1$, where Q_1 is the first quartile and Q_3 the third quartile. In particular, the lower-half interquartile range is defined as the difference between the median and the lower quartiles $IQR_L = median - Q_1$. Similarly, the upper-half interquartile range is defined as the difference between the upper quartile and the median $IQR^U = Q_3 - median$.

The ratio between these two concepts provides an estimation of the skewness of the arc frequency distribution curve (we remind the reader that the arc frequency distribution curve is defined in the range [0, 1]), where $\rho_{IQR} = \frac{IQR^U}{IQR_L} > 1$ indicates a positively skewed distribution.

Using this ratio, the best frequency threshold is the threshold that removes the minimum amount of infrequent arcs (i.e. arcs with a frequency below the threshold) producing an arc frequency distribution curve where $\rho_{IQR} \leq 1$. Formally, ε can be defined as a value $x \in [0, \Lambda_\lambda(\rightsquigarrow_x^m)]$ for which:

$$\rho_{IQR}(\rightsquigarrow_x^m) \leq 1 \wedge \nexists y \in [0,1][\rho_{IQR}(\rightsquigarrow_y^m) \leq 1 \wedge |\rightsquigarrow_x^m| > |\rightsquigarrow_y^m|],$$

where $\rightsquigarrow_x^m$ is the set of frequent arcs obtained using x as a threshold, $\rho_{IQR}(\rightsquigarrow_x^m)$ is the ρ_{IQR} measured over the set of arcs identified by $\rightsquigarrow_x^m$, and $\Lambda_\lambda(\rightsquigarrow_x^m)$ is the value of the λ percentile of the arcs frequencies measured over the set of arcs identified by $\rightsquigarrow_x^m$.

When infrequent arcs, and consequently events, are removed, the frequencies of the other arcs change, which affects the arc frequency distribution curve. In order to address this problem we propose to reiterate the log filtering several times using as input the filtered log, until no more events are removed.

The log filtering technique presented in Sections 3–5 is described in pseudo-code in Algorithm 1.

Algorithm 1: FilterLog

input: Event log $\mathscr{L}$, finite set of tasks Γ, finite set of required tasks $\Gamma^R \subseteq \Gamma$, percentile λ

DirectFollowDependencies $\rightsquigarrow \Leftarrow$ computeDFD($\mathscr{L}$);
LogAutomaton $\mathscr{A} \Leftarrow$ generateAutomaton(Γ, $\rightsquigarrow$);
FrequencyThreshold $\varepsilon \Leftarrow$ discoverFrequencyThreshold($\rightsquigarrow$, λ);
FrequentArcs $\rightsquigarrow^m \Leftarrow$ discoverFrequentArcs($\rightsquigarrow$, ε);
InfrequentArcs $\rightsquigarrow^o \Leftarrow \rightsquigarrow \setminus \rightsquigarrow^m$;
AnomalyFreeAutomaton $\mathscr{A}^f \Leftarrow$ solveILPProblem($\mathscr{A}$, Γ, Γ^R, $\rightsquigarrow^m$, $\rightsquigarrow^o$);
PetriNet *petrinet* $\Leftarrow$ convertAutomaton($\mathscr{A}^f$);
Alignment *alignment* $\Leftarrow$ alignPetriNetWithLog(*petrinet*, $\mathscr{L}$);
FilteredLog $\mathscr{F} \Leftarrow$ removeNonReplayableEvents($\mathscr{L}$, *alignment*);
if $|\mathscr{F}| < |\mathscr{L}|$ **then**
  $\mathscr{L} \Leftarrow \mathscr{F}$;
  go to 1;
return $\mathscr{F}$

6 Evaluation

In this section we present the results of three experiments to assess the goodness of our filtering technique. To perform these experiments, we implemented the technique as a plugin, namely the "Infrequent Behavior Filter" plugin, for the ProM framework[7]. The plug-in can use either Gurobi[8] or LPSolve[9] as ILP solver.

To identify infrequent events we used the *alignment-based replay* technique proposed in [3]. This technique replays a log and a Workflow net simultaneously, and at each state it identifies one of three types of move: "synchronous move", when an event is executed in the log in synch with a transition in the net, "move on log", when an event executed in the log cannot be mimicked by a corresponding transition in the net, and "move on model" vice-versa. To apply this technique, we convert a log automaton into a Workflow net, i.e. a Petri net with a single source and a single sink. The conversion is obtained by creating a transition with a single incoming place and a single outgoing place for each state, while each arc in the automaton is converted into a silent transition connecting the outgoing place of the source of the arc with the incoming place of the target of the arc. In case of multiple initial states a fictitious place is introduced. This place is connected to every initial state via a silent transition for each initial state. Similarly, in case of multiple final states, an artificial place is introduced and every final state is connected to the new place via a silent transition for each final state. The obtained Workflow net is then aligned with the log. In order to remove infrequent events from the log we have to remove all events corresponding to a "move on log", i.e. those events that exist in the log but cannot be reproduced by the automaton. We decided to use the replay-based alignment as it guarantees optimality under the assumption that the Workflow net is *easy sound* [1]. A Workflow net is easy sound if there is at least one firing sequence that can reach the final marking [34]. This condition is fulfilled by construction since the Workflow net obtained from an automaton has at least one execution path from source to sink and does not contain transitions having multiple incoming or outgoing arcs (i.e. there is no concurrency).

6.1 Design

The design for the three experiments is illustrated in Figure 6. The first two experiments were aimed at measuring how our technique copes with infrequent behavior in a controlled environment. For this we used artificially generated logs where we incrementally injected infrequent behavior. The third experiment, performed on real-life logs, aimed at verifying if the same levels of performance can be achieved in a real-life scenario.

In the first experiment, starting from an artificial log we generated several logs by injecting different levels of infrequent behavior. These logs were provided as input to our filtering technique. We then measured the amount of infrequent behavior correctly identified by our technique, by computing the *sensitivity* (recall) and the *positive predictive value* (precision) of our technique.[10]

In the second experiment the artificial logs previously generated were provided as input to several baseline discovery algorithms, before and after applying our filtering technique. We then measured the quality of the discovered models against the original log in terms of fitness, precision, generalization and complexity using the metrics described in Section 2. Additionally, as part of this second experiment we also compared our filtering technique with the SLF and PCL filtering techniques described in Section 2.

Finally, in the third experiment, we compared the results of the baseline discovery algorithms before and after applying our filtering technique using various real-life logs. In this last experiment, we did not consider other filtering techniques since in the second experiment we demonstrated the advantages of our technique over existing ones. In addition, we measured the time performance of our technique when filtering out infrequent behavior from real-life logs.

For the second and third experiment, we used the following discovery algorithms: InductiveMiner [23], Heuristics Miner [38], Fodina [36] and ILP Miner [35]. We excluded the Fuzzy Miner since fuzzy models do not have a well-defined semantics. For each of these algorithms we used the default settings, since we were interested in the relative improvement of the discovery result and not in the absolute value.

In all experiments we used the following settings. We set the λ percentile to 0.125 (i.e. half of a quartile), to allow a fine-tuned removal of events in multiple small steps, selected all activities as required, and used these settings to discover the frequency threshold in order to filter out infrequent order dependencies from the automaton. We used Guribi as the ILP solver.

The results of these experiments, as well as the artificial datasets that we generated, are provided with the software distribution.

6.2 Datasets

For the first experiment, we generated a base log using CPN Tools[11] and from this base log we produced two sets of eight "noisy" logs by injecting an incremental amount of infrequent behavior, as a percentage of its total number of events. We used different percentages ranging from 5% to 40%, with increments of 5%, in order to simulate various levels of infrequent behavior in real-life logs (we stopped at 40% since above this threshold the behavior is no longer infrequent).

The first logset was obtained by inserting additional events in the logs, while the second logset was obtained by inserting as well as removing events (in equal amount). The labels of these events were selected in such a way that the insertion or removal of

7. Available at https://svn.win.tue.nl/trac/prom/browser/Packages/NoiseFiltering
8. http://www.gurobi.com
9. http://lpsolve.sourceforge.net
10. We used the terms *sensitivity* and *positive predictive value* to avoid confusion with discovery fitness and precision.
11. http://cpntools.org

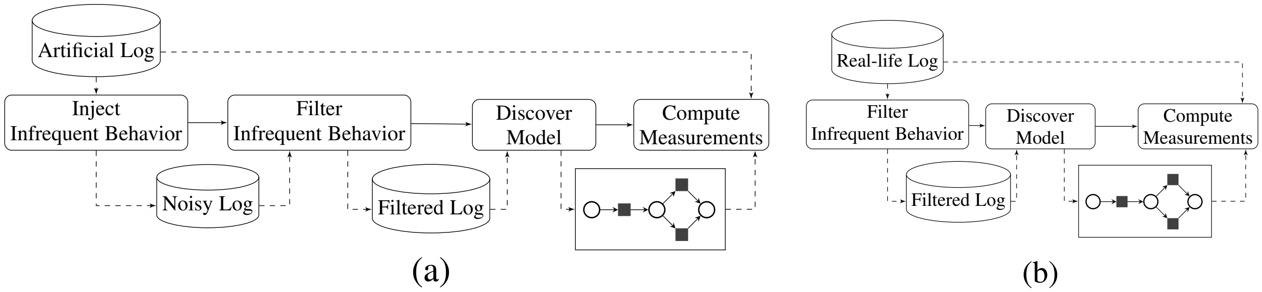


Fig. 6: Experimental setup for artificial (Fig 6a) and real-life logs (Fig 6b).

an event did not yield a direct follow dependency that was already present in the original log. We used a uniform distribution to select in which traces and at which positions in those traces events were to be inserted or removed.

For the second experiment we used the first of these two logsets, i.e. that where infrequent behavior is captured by additional events only. This is because we wanted to measure the accuracy of our technique against the approach upon which this has been designed, which is that of filtering out additional infrequent behavior.

For the third experiment, however, we used four real-life logs from different domains and of different size, for which we do not have insights on the type of infrequent behavior. We used these logs to evaluate the generalizability of the results obtained with the first two experiments. Specifically, we used logs from financial and medical institutions, and from Australian and Dutch companies. Two such logs are publicly available and are those used for the 2012[12] and 2014[13] editions of the BPI Challenge. These two logs were pre-filtered by removing infrequent labels (using the SLF filter with a threshold of 90%). Using the full log was not possible due to timing issues. For example, the measurement of the fitness over a structured model produced using Inductive Miner using as input the BPI Challenge 2012 log took over 30 minutes.

Table 2 reports the characteristics of all logs used in terms of number of traces, number of events, number of unique labels for each log, and percentage of infrequent behavior. The latter is the percentage of events added for the artificial logs, and the percentage of events removed from the real-life logs, given that for real-life logs we did not have an infrequent behavior-free version. In total we have a variety of logs ranging from a minimum of 617 traces to a maximum of 46,616 traces, from a minimum of 9,575 events to a maximum of 422,563 events, from a minimum of 9 labels to a maximum of 22 labels. Likewise, the level of infrequent behavior observed in real-life logs varies from 2% to 39%.

6.3 Results

As for the first experiment, Figure 7 plots the sensitivity and the positive predictive value of our technique in identifying and removing infrequent behavior, while varying the level of infrequent behavior in the log, for both logsets. Sensitivity is 0.9 independently of the level of infrequent behavior and of its type (i.e. additional and/or missing events). The positive predictive value, on the other side, never drops below 0.74 when the logs only contain additional events, while it fluctuates when events are

12. doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
13. doi:10.4121/uuid:86977bac-f874-49cf-8337-80f26bf5d2ef

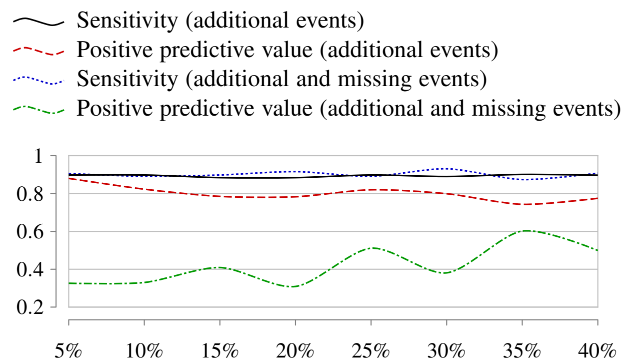


Fig. 7: Sensitivity and positive predictive value of our filtering technique with varying levels of infrequent behavior, with additional events, as well as with additional and missing events.

missing from the log. This is due to the fact that our technique cannot insert missing events when fixing the log, causing entire traces to be discarded.

When analysing these measures we need to keep in mind that logs with high levels of infrequent behavior (e.g. a level above 40%) pose a contradiction, as a high level of infrequent behavior essentially corresponds to frequent behavior. We can observe that our technique can correctly identify infrequent behavior, being accurate as long as the amount of infrequent behavior is below 40% of the total number of events.

Coming to the second experiment, Figure 8 shows the results of fitness, precision, F-score, and model size dimensions (cf. Section 2.4 for the explanation of each metric), obtained through application of the baseline discovery algorithms, with and without our filtering technique, on the artificial logs. From these results we can draw a number of observations. First, Heuristics Miner and Fodina present a drop in precision (with a consequent drop in the F-score value) and an increase in size when the amount of noise increases, despite being noise-tolerant. This behavior cannot be observed in the models discovered by the InductiveMiner and the ILP Miner which can keep a constant level of F-score despite increasing levels of noise. However, as a side effect, the precision achieved by these two algorithms is very low (stable at around 0.2), which determines the low level of F-score (around 0.3 for InductiveMiner and around 0.25 for ILP Miner).

Second, and most importantly, the results confirm the effectiveness of our technique. The F-score significantly improves when our technique is used compared to when it is not used (Mdn 0.892 instead of 0.320, with Mann-Whitney test U = 56, z = -6.139, $p = 0.000 < 0.05$). This significant increment is explained by the noticeable and significant increment of precision (Mdn 0.815 instead of 0.189, with Mann-Whitney test U = 64, z = -6.031, p = 0.000 $<$ 0.05) and small but significant increment of fitness (Mdn 0.996 instead of 0.922, with Mann-Whitney test U = 288, z = -3.061, p = 0.002 $<$ 0.05). Such an increment of F-score is less noticeable for models generated by the ILP Miner. This is because the ILP miner, in order to fit every trace into the model, is prone to generate "flower" models which have high fitness but suffer from low precision.

Third, our technique also reduces the complexity of the discovered models in a statistically significant way. Before its application, the discovered model has a median of 69 nodes, which is reduced to 49.5 after the application (Mann-Whitney test U = 872, z = 4.843, p = 0.000 $<$ 0.05). The Appendix reports the measurements of the other structural complexity metrics: the decrease in CFC, ACD and CNC confirm the reduction in complexity observed from the results on model size in Figure 8. The increase in density is expected, as this metric is inversely correlated with size (smaller models tend to be denser) [24].

Finally, our technique improves fitness, precision, and reduces complexity without negatively affecting generalization. In fact, the latter does not vary and remains constant to a median of 0.997.

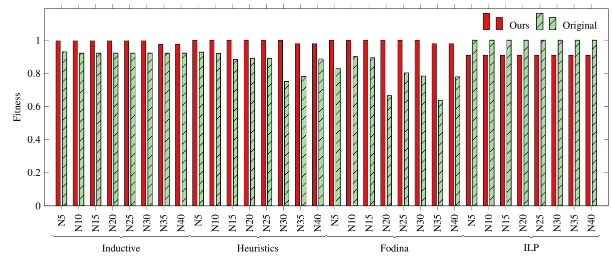
Additionally, we compared our technique with the SLF and the PCL filtering techniques presented in Section 2.[14] Figure 9 shows the results of fitness, precision, F-score, and model size dimensions, obtained when applying all three techniques. The results refer to the eight artificial logs used in the previous experiment, and are averaged across all the discovery algorithms used before.

Fitness differs significantly among the three techniques (Ours: Mdn 0.996, SLF: Mdn 0.848, and PCL: Mdn 0.886, with Kruskal-Wallis test H(2) = 44.351, p = 0.000 $<$ 0.05). due to a significant improvement of our technique over the other two (Fitness Dunn-Bonferroni post hoc analysis: SFL-VS-Ours $H(1) = 44.375$, $p = 0.000$; PCL-VS-Ours $H(1) = 33.625$, $p = 0.000$).

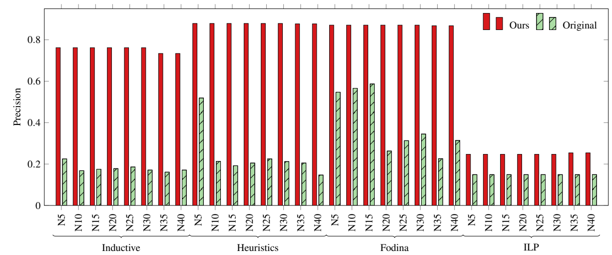
14. Standard paramenters were used for SLF and PCL.

Artificial Log	#Traces	#Events	#Unique Labels	%Infrequent Behavior
N5$_a$	2249	13559	13	5%
N10$_a$	2249	14312	13	10%
N15$_a$	2249	15154	13	15%
N20$_a$	2249	16101	13	20%
N25$_a$	2249	17175	13	25%
N30$_a$	2249	18401	13	30%
N35$_a$	2249	19817	13	35%
N40$_a$	2249	21468	13	40%
N5$_{a\&m}$	2249	12881	13	5%
N10$_{a\&m}$	2249	12880	13	10%
N15$_{a\&m}$	2249	12880	13	15%
N20$_{a\&m}$	2249	12881	13	20%
N25$_{a\&m}$	2249	12881	13	25%
N30$_{a\&m}$	2249	12881	13	30%
N35$_{a\&m}$	2249	12881	13	35%
N40$_{a\&m}$	2249	12880	13	40%
Real-life Log	**#Traces**	**#Events**	**#Unique Labels**	**%Infrequent Behavior**
BPI2012	13087	148192	15	39%
BPI2014	46616	422563	9	13%
Hospital1	688	9575	19	2%
Hospital2	617	9666	22	2%

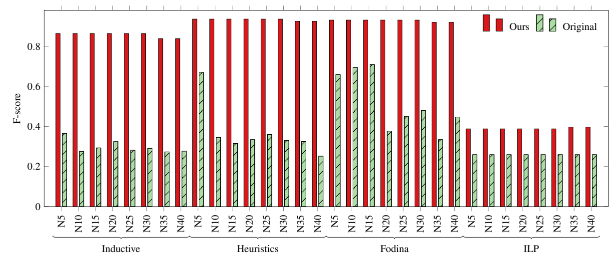
TABLE 2: Characteristics of the logs used in the evaluation.



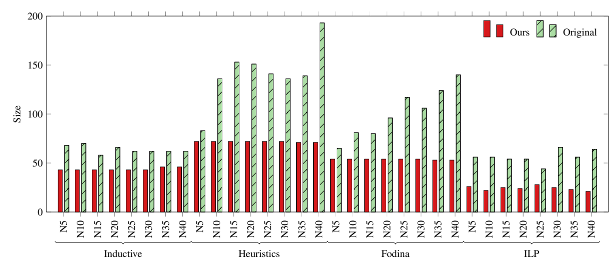
(a) Fitness comparison on artificial logs



(b) Precision comparison on artificial logs



(c) F-score comparison on artificial logs



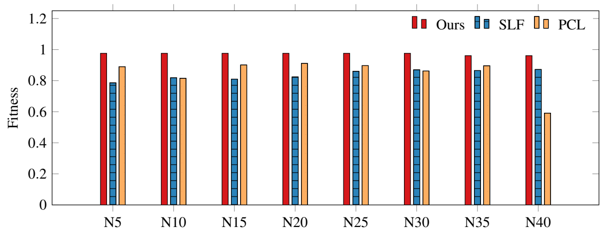
(d) Size comparison on artificial logs

Fig. 8: Fitness, Precision, F-score, and Size comparison between filtered and original log using different artificial logs and discovery algorithms.

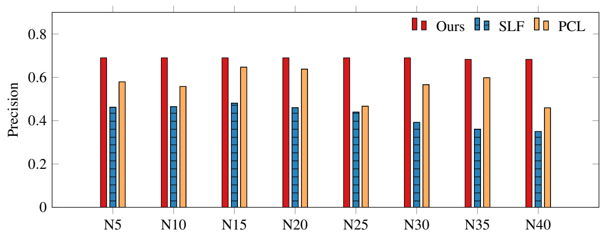
Precision also presents significant differences among the three techniques (Ours: Mdn 0.815, SLF: Mdn 0.410, and PCL: Mdn 0.644, with Kruskal-Wallis test H(2) = 18.005, p = 0.000 $<$ 0.05). This difference is explained by a significant improvement of our technique over the SLF filter (Precision Dunn-Bonferroni post hoc analysis: SFL-VS-Ours $H(1) = 29.531$, $p = 0.000$). As shown in Figure 9.d, our technique has also a noticeable improvement over PCL, though this is not statistically significant.

As expected, these differences in terms of fitness and precision are reflected on the F-score (Ours: Mdn 0.892, SLF: Mdn 0.559, and PCL: Mdn 0.746, with Kruskal-Wallis test H(2) = 20.123, p = 0.000 $<$ 0.05), with a significant improvement of our technique over the other two (F-score Dunn-Bonferroni post hoc analysis: SLF-VS-Ours Kruskal-Wallis test $H(1) = 30.938$, $p = 0.000$; PCL-VS-Our Kruskal-Wallis test $H(1) = 19.125$, $p = 0.006$).

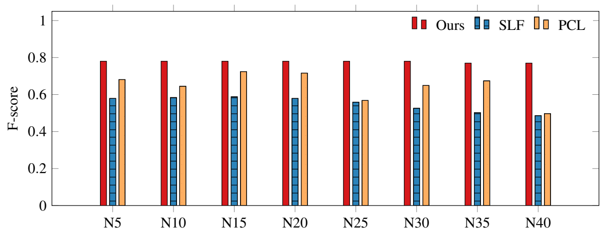
Finally, there are also significant differences in terms of size (Ours: Mdn 49.5, SLF: Mdn 36.5, and PCL: Mdn 51, with



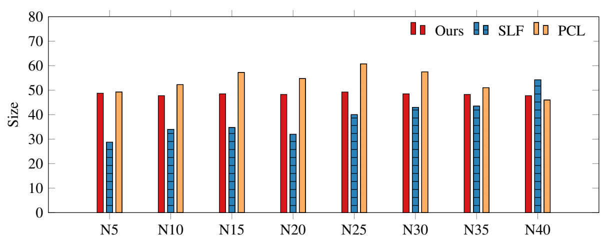
(a) Fitness comparison on artificial logs



(b) Precision comparison on artificial logs



(c) F-score comparison on artificial logs



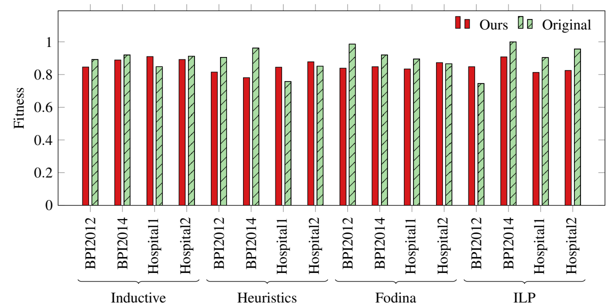
(d) Size comparison on artificial logs

Fig. 9: Fitness, Precision, F-score, and Size using three different log filtering techniques: ours, SLF and PCL, for different artificial logs and averaged across different discovery algorithms.

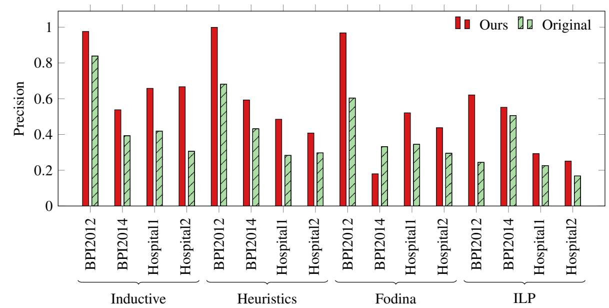
Kruskal-Wallis test H(2) = 11.473, p = 0.003 $<$ 0.05). Our technique leads to models that are generally smaller than those produced by PCL, though the use of SLF produces the smallest models. This is because this filter removes all events of infrequent activities, leading to a significantly lower number of nodes (Size Dunn-Bonferroni post hoc analysis: SFL-VS-PCL Kruskal-Wallis test $H(1) = -23.188$, $p = 0.001$).

The results on real-life logs, summarized in Figure 10, are in line with those obtained on artificial logs. The F-score improves (Mdn 0.673 instead of 0.493) due to a significant improvement in precision (Mdn 0.545 instead of 0.339, with Mann-Whitney test U = 72.0, z = -2.111, p = 0.035 $<$ 0.05) despite a reduction in fitness (Mdn 0.847 instead of 0.905, with Mann-Whitney test U = 196.5, z = 2.583, p = 0.008 $<$ 0.05). The size of the discovered model is again significantly reduced from a median of 91 elements to a median of 64.5 elements (Mann-Whitney test U = 182, z = 2.036, p = 0.043 $<$ 0.05). Similarly, in the Appendix

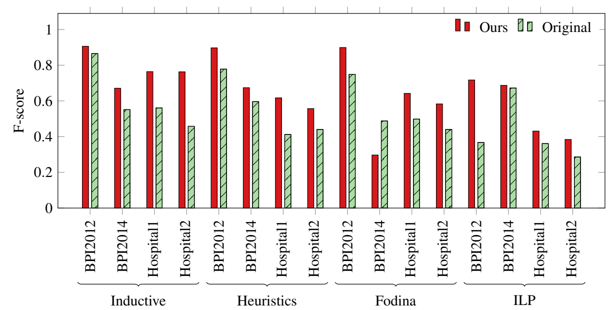
15. The drop in precision (and F-score) in the result obtained from Fodina on the BPI2014 log is caused be the discovered model being unsound.



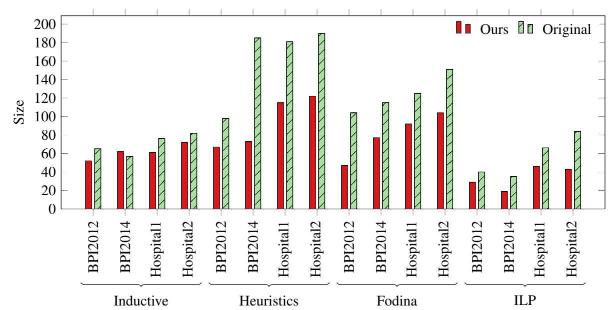
(a) Fitness comparison on real-life logs



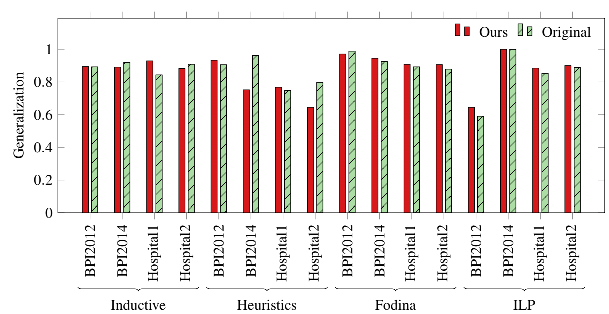
(b) Precision comparison on real-life logs



(c) F-score comparison on real-life logs



(d) Size comparison on real-life logs



(e) Generalization comparison on real-life logs

Fig. 10: Fitness, Precision, F-score, Size and Generalization comparison between filtered and original log using different real-life logs and discovery algorithms.[15]

Log	Filtering?	Filtering time		Discovery time							
				Inductive		Heuristics		Fodina		ILP	
		Avg [secs]	StDev [secs]	Avg [secs]	StDev [secs]	Avg [secs]	StDev [secs]	Avg [secs]	StDev [secs]	Avg [secs]	StDev [secs]
BPI 2012	Yes	249.1	7.26	1.5	0.09	1.7	0.09	2.1	0.21	146.5	3.74
	No	-	-	2.1	0.05	2.3	0.04	5.7	0.81	292.1	8.90
BPI 2014	Yes	812.4	24.39	4.6	0.11	5.6	0.23	15.2	0.56	161.2	1.12
	No	-	-	6.9	0.34	7.5	0.23	49.6	13.56	714.7	28.74
Hospital 1	Yes	31.6	1.5	0.5	0.03	0.4	0.04	0.4	0.07	151.5	2.57
	No	-	-	0.6	0.08	0.4	0.05	0.6	0.02	153.6	2.09
Hospital 2	Yes	49.7	2.21	0.5	0.09	0.4	0.03	0.4	0.03	161.7	1.56
	No	-	-	0.5	0.04	0.4	0.09	0.5	0.08	163	2.13

TABLE 3: Time performance on real-life logs, with and without using our filtering technique.

we can see that CFC, ACD, CNC decrease also for the real-life logs. Finally, generalization slightly increases from a median of 0.892 to a median of 0.897.

Time performance. In the experiment with real-life logs, the technique took on average 286 secs to filter a log, with a minimum time of 30 secs (Hospital 1) and a maximum time of 14.23 mins (BPI 2014). The summary statistics of the time performance obtained for every log (computed over 10 runs) are reported in Table 3. As we can see, time performance is within reasonable bounds. Moreover, the additional time required for filtering a log is compensated by the time saved afterwards, in the discovery of the process model. On average, model discovery is 1.7 times faster when using a filtered log.

7 Conclusion

In this paper we presented a technique for the automatic removal of infrequent behavior from process execution logs. The core idea is to use infrequent direct follows dependencies between event labels as a proxy for infrequent behavior. These dependencies are detected and removed from an automaton built from the event log, and then the original log is updated accordingly, by removing individual events using alignment-based replay [3].

We demonstrated the effectiveness and efficiency of the proposed technique using a variety of artificial and real-life logs, on top of mainstream process discovery algorithms. The results show a significant improvement over fitness, precision and complexity without a negative effect on generalization. Time performance is within reasonable bounds and model discovery is on overage 1.7 times faster when using the log filtered with our technique. Moreover, the comparison with two baseline techniques for automatic filtering of process event logs, shows that our technique provides a statistically significant improvement of fitness and precision over these techniques, while leading to models of comparable size. These improvements are a byproduct of a noise-free log. A noise-free log contains less events and direct-follow dependencies. These two elements play a significant role on the performance and accuracy of a discovery algorithm. The performance of a discovery algorithm is proportional to the number of events in the log. Hence, less events in the log means less time required to discover a model. Additionally, less direct-follow dependencies means less (infrequent) behavior that a model should account for, hence the improvement in accuracy (fitness and precision) as well as in model complexity.

In future, we plan to consider other types of event dependencies, e.g. transitive ones, as well as to improve the handling of logs with missing events. Another avenue for future work is to develop a technique to isolate behavior (frequent or infrequent) in order to compare logs with similar behaviors.

Acknowledgment

This research is funded by the ARC Discovery Project DP150103356, and supported by the Australian Centre for Health Services Innovation #SG00009-000450.

References

[1] A. Adriansyah. *Aligning Observed and Modeled Behaviour*. PhD thesis, Technische Universiteit Eindhoven, 2014.

[2] A. Adriansyah, J. Munoz-Gama, J. Carmona, B.F. van Dongen, and W.M.P. van der Aalst. Alignment based precision checking. In *Proc. of BPM Workshops*, pages 137–149, 2012.

[3] A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Conformance checking using cost-based fitness analysis. In *Proc. of EDOC*, pages 55–64, 2011.

[4] C.C. Aggarwal. *Outlier Analysis*. Springer, 2013.

[5] S. Basu and M. Meckesheimer. Automatic outlier detection for time series: an application to sensor data. *KAIS*, 11(2):137–154, 2006.

[6] S. Budalakoti, A.N. Srivastava, and M.E. Otey. Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety. *IEEE TSMCS*, 39(1):101–113, Jan 2009.

[7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection for discrete sequences: A survey. *IEEE TKDE*, 24(5):823–839, May 2012.

[8] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa. Beyond tasks and gateways: Discovering BPMN models with subprocesses, boundary events and activity markers. In *Proc. of BPM*, pages 101–117, 2014.

[9] K. Das, J. Schneider, and D.B. Neill. Anomaly pattern detection in categorical datasets. In *Proc. of ACM SIGKDD*, pages 169–176, 2008.

[10] G. Florez-Larrahondo, S.M. Bridges, and R. Vaughn. Efficient modeling of discrete events for anomaly detection using hidden markov models. In *Proc. of ISC*, pages 506–514, 2005.

[11] C.W. Günther and W.M.P. van der Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In *Proc. of BPM*, pages 328–343, 2007.

[12] M. Gupta, C.C. Aggarwal, and J. Han. Finding top-k shortest path distance changes in an evolutionary network. In *Proc. of SSTD*, pages 130–148. Springer, 2011.

[13] M. Gupta, J. Gao, C.C. Aggarwal, and J. Han. Outlier detection for temporal data: A survey. *IEEE TKDE*, 26(9):2250–2267, 2014.

[14] M. Gupta, A. Mallya, S. Roy, J.H.D. Cho, and J. Han. *Local Learning for Mining Outlier Subgraphs from Network Datasets*, pages 73–81. 2014.

[15] R. Gwadera, M.J. Atallah, and W. Szpankowski. Reliable detection of episodes in event sequences. *KAIS*, 7(4):415–437, May 2005.

[16] S.A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *J. Comput. Secur.*, 6(3):151–180, August 1998.

[17] R.M. Karp. Reducibility among combinatorial problems. In *Proc. of CCC*, pages 85–103. Springer US, 1972.

[18] E. Keogh, J. Lin, S.-H. Lee, and H. van Herle. Finding the most unusual time series subsequence: algorithms and applications. *KAIS*, 11(1):1–27, 2006.

[19] E. Keogh, S. Lonardi, and B. Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proc. of ACM SIGKDD*, pages 550–556, 2002.

[20] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. of IJCAI*, pages 1137–1145, 1995.
[21] T. Lane and C.E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *Proc of AI*, 1997.
[22] T. Lane and C.E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM TISS*, 2(3):295–331, 1999.
[23] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *Proc. of BPM Workshops*, pages 66–78, 2014.
[24] J. Mendling, H.A. Reijers, and J. Cardoso. What makes process models understandable? In *Proc. of BPM*, pages 48–63, 2007.
[25] S. Muthukrishnan, R. Shah, and J.S. Vitter. Mining deviants in time series data streams. In *Proc. of SSDM*, pages 41–50, 2004.
[26] Object Management Group (OMG). *Business Process Model and Notation (BPMN) ver. 2.0*. Object Management Group (OMG), January 2011.
[27] P. Sun, S. Chawla, and B. Arunasalam. Mining for outliers in sequential databases. In *Proc. of SIAM*, pages 94–105, 2006.
[28] S. Suriadi, R. Andrews, A.H.M. ter Hofstede, and M. Wynn. Event log imperfection patterns for process mining - towards a systematic approach to cleaning event logs. *Information Systems*, July 2016.
[29] S. Suriadi, R. Mans, M. Wynn, A. Partington, and J. Karnon. Measuring patient flow variations: A cross-organisational process mining approach. In *Proc. of AP-BPM*, pages 43–58, 2014.
[30] S. Suriadi, M. Wynn, C. Ouyang, A.H.M. ter Hofstede, and N.J. van Dijk. Understanding process behaviours in a large insurance company in australia: A case study. In *Proc. of CAiSE*, pages 449–464, 2013.
[31] W.M.P. van der Aalst. *Process Mining - Data Science in Action*, volume 2nd Edition. Springer, 2016.
[32] W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
[33] W.M.P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE TKDE*, 16(9):1128–1142, 2004.
[34] R.A. van der Toorn. *Component-based software design with Petri nets: an approach based on inheritance of behavior*. PhD thesis, Technische Universiteit Eindhoven, 2004.
[35] J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. *Fundam. Inform.*, 94(3-4):387–412, 2009.
[36] S.K.L.M. vanden Broucke, J. De Weerdt, J. Vanthienen, and B. Baesens. Fodina: a robust and flexible heuristic process discovery technique. http://www.processmining.be/fodina/, 2013. Last accessed: 03/27/2014.
[37] J. Wang, S. Song, X. Lin, X. Zhu, and J. Pei. Cleaning structured event logs: A graph repair approach. In *Proc. of ICDE*, pages 30–41, 2015.
[38] A.J.M.M. Weijters and J.T.S. Ribeiro. Flexible heuristics miner (FHM). In *Proc. of CIDM*, pages 310–317, 2011.
[39] D. Yankov, E. Keogh, and U. Rebbapragada. Disk aware discord discovery: finding unusual time series in terabyte sized datasets. *KAIS*, 17(2):241–262, 2008.
[40] X. Zhang, P. Fan, and Z. Zhu. A new anomaly detection method based on hierarchical hmm. In *Proc. of PDCAT*, pages 249–252, 2003.

Raffaele Conforti is Research Fellow in the School of Information Systems at the Queensland University of Technology, Australia. He is conducting research in the area of business process management, with respect to business process automation and process mining. In particular his research focuses on automated process discovery and automatic detection, prevention and mitigation of process-related risks during the execution of business processes.

Marcello La Rosa is Professor of Business Process Management (BPM) and the academic director for corporate programs and partnerships at the Information Systems school of the Queensland University of Technology, Brisbane, Australia. His research interests include process consolidation, mining and automation, in which he published over 80 papers. He leads the Apromore initiative (www.apromore.org) a strategic collaboration between various universities for the development of an advanced process model repository. He is co-author of the textbook Fundamentals of Business Process Management (Springer, 2013).

Arthur H.M. ter Hofstede is a Professor in the Information Systems School in the Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia, and is Head of the Business Process Management Discipline. He is also a Professor in the Information Systems Group of the School of Industrial Engineering of Eindhoven University of Technology, Eindhoven, The Netherlands. His research interests are in the areas of business process automation and process mining.

APPENDIX

Table 4 reports the measurements of the structural complexity metrics: CFC, ACD, CNC, and density. The decrease in CFC, ACD and CNC confirm the reduction in complexity observed from the results on model size in Figure 8. The increase in density is expected, as this metric is inversely correlated with size (smaller models tend to be denser) [24].

Log	Measure	Inductive Original	Inductive Ours	Heuristics Original	Heuristics Ours	Fodina Original	Fodina Ours	ILP Original	ILP Ours
N5	CFC	39	**20**	65	**31**	45	**31**	205	**66**
	ACD	**3.500**	3.818	4.182	**3.444**	3.840	**3.444**	14.093	**9.190**
	CNC	1.294	**1.209**	1.422	**1.167**	1.338	**1.222**	6.927	**3.885**
	Density	**0.019**	0.029	0.017	**0.016**	**0.021**	0.023	**0.128**	0.155
N10	CFC	41	**20**	188	**31**	65	**31**	212	**37**
	ACD	**3.533**	3.818	7.911	**3.444**	4.333	**3.444**	14.345	**6.467**
	CNC	1.314	**1.209**	2.949	**1.167**	1.506	**1.222**	7.054	**2.500**
	Density	**0.019**	0.029	0.022	**0.016**	**0.019**	0.023	0.128	**0.119**
N15	CFC	35	**20**	221	**31**	64	**31**	193	**63**
	ACD	3.917	**3.818**	8.532	**3.444**	4.278	**3.444**	13.453	**9.050**
	CNC	1.379	**1.209**	3.320	**1.167**	1.500	**1.222**	6.611	**3.800**
	Density	**0.024**	0.029	0.022	**0.016**	**0.019**	0.023	**0.125**	0.158
N20	CFC	39	**20**	216	**31**	105	**31**	202	**56**
	ACD	3.643	**3.818**	8.057	**3.444**	5.893	**3.444**	14.132	**8.263**
	CNC	1.333	**1.209**	3.119	**1.167**	2.125	**1.222**	6.944	**3.458**
	Density	**0.021**	0.029	0.021	**0.016**	**0.022**	0.023	**0.131**	0.150
N25	CFC	37	**20**	196	**31**	129	**31**	166	**76**
	ACD	**3.769**	3.818	7.188	**3.444**	5.765	**3.444**	14.070	**9.783**
	CNC	1.355	**1.209**	2.759	**1.167**	2.085	**1.222**	6.886	**4.179**
	Density	**0.022**	0.029	0.020	**0.016**	**0.018**	0.023	0.160	**0.155**
N30	CFC	37	**20**	185	**31**	106	**31**	230	**57**
	ACD	**3.769**	3.818	6.967	**3.444**	5.309	**3.444**	13.246	**7.850**
	CNC	1.355	**1.209**	2.654	**1.167**	1.849	**1.222**	6.530	**3.320**
	Density	**0.022**	0.029	0.020	**0.016**	**0.018**	0.023	**0.101**	0.138
N35	CFC	37	**22**	191	**30**	143	**30**	187	**37**
	ACD	**3.769**	3.833	7.085	**3.333**	6.842	**3.333**	12.527	**6.267**
	CNC	1.355	**1.217**	2.712	**1.155**	2.476	**1.208**	6.161	**2.348**
	Density	**0.022**	0.027	0.020	**0.017**	**0.020**	0.023	0.112	**0.107**
N40	CFC	37	**22**	301	**30**	152	**30**	232	**28**
	ACD	**3.769**	3.833	8.527	**3.333**	6.481	**3.333**	13.794	**5.667**
	CNC	1.355	**1.217**	3.497	**1.155**	2.257	**1.208**	6.797	**2.000**
	Density	**0.022**	0.029	0.018	**0.017**	**0.016**	0.023	0.108	**0.100**
BPI 2012	CFC	34	**29**	66	**21**	71	**25**	118	**18**
	ACD	**3.273**	3.333	4.516	**3.143**	5.407	**3.250**	11.686	**4.500**
	CNC	1.200	**1.193**	1.388	**1.101**	1.433	**1.164**	5.225	**1.429**
	Density	**0.019**	0.021	**0.014**	0.016	**0.014**	0.022	0.134	**0.053**
BPI 2014	CFC	**25**	26	438	**64**	131	**57**	114	**37**
	ACD	3.556	**3.100**	8.183	**5.606**	5.030	**4.538**	12.147	**7.267**
	CNC	1.228	**1.161**	3.551	**1.795**	1.861	**1.416**	5.914	**3.167**
	Density	0.022	**0.019**	**0.019**	0.025	**0.016**	0.019	**0.174**	0.186
Hospital 1	CFC	39	**22**	300	**149**	124	**33**	337	**47**
	ACD	**3.556**	4.889	6.708	**5.115**	4.552	**3.778**	17.734	**7.647**
	CNC	**1.263**	1.300	2.464	**1.681**	1.584	**1.254**	8.621	**2.741**
	Density	**0.017**	0.033	0.014	**0.012**	**0.013**	0.022	0.133	**0.105**
Hospital 2	CFC	38	**26**	308	**41**	137	**39**	571	**64**
	ACD	**3.615**	4.000	6.836	**4.190**	4.206	**4.000**	26.728	**9.273**
	CNC	**1.244**	1.277	2.395	**1.247**	1.490	**1.306**	12.917	**3.548**
	Density	**0.015**	0.028	**0.013**	0.014	**0.010**	0.021	0.156	**0.118**

TABLE 4: Structural complexity measurements for the second and the third experiment.