Filtering out Infrequent Behavior from Process Event Logs

Raffaele Conforti*, Marcello La Rosa*† and Arthur H.M. ter Hofstede*‡
Queensland University of Technology, Australia
Email: {raffaele.conforti, m.larosa, a.terhofstede}@qut.edu.au
‡NICTA Queensland Lab, Australia
†Eindhoven University of Technology, The Netherlands

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 approach to the removal of infrequent behavior from event logs. The proposed approach 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.

I. INTRODUCTION

Process mining aims to extract actionable process knowledge from event logs of IT systems that are commonly available in contemporary organizations [1]. 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 [2].

Algorithms for process discovery operate on the assumption that an event log faithfully represents the behavior of a 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. In process event logs these outliers represent infrequent behavior (often referred to as “noise”) [2], [3] 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, thus cluttering the model, or that are simply not a true representation of actual behavior.

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 simplicity. 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 [4], Fodina [5], and Inductive Miner [6], 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 2% of the total log size.

This paper proposes a technique for systematically filtering infrequent behavior from process execution logs. First an abstraction is built in the form of an automaton (a directed graph) which captures the direct follows dependencies between activity labels as found 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 proposed 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 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. First, improvement of accuracy and reduction of complexity in the presence of varying levels of noise was evaluated for a number of baseline process discovery algorithms. This evaluation use artificially generated logs to control the levels of noise. The same kind of evaluation was then repeated using a variety of real-life logs exhibiting different characteristics such as overall size and number of (distinct) events. Accuracy was measured in terms of the well-established measures of recall (i.e. fitness) and precision (i.e. behavioral appropriateness), 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 recall, precision, and complexity while the generality of the discovered model is not negatively affected. As an example, Fig. 1 shows two BPMN models, one obtained by applying the Inductive Miner to the log of an Australian hospital and one obtained by applying the Inductive Miner to this log after it was filtered with the
The proposed technique.\textsuperscript{1} Time performance tests show that the technique scales well to large and complex logs, being able to preprocess a log generally in few minutes.

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

\section{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.

\textsuperscript{1}Activity labels have been masked to preserve anonymity of the hospital.

\textsuperscript{2}Different labels between the two models are due to the events filtering of InductiveMiner.

\textsuperscript{3}http://www.processmining.org/

A. Process Log Filtering

Preprocessing a log before starting any type of process mining exercise is a de-facto practice. In general this preprocessing includes a log filtering phase. While several plug-ins for log filtering are available as part of the ProM Framework\textsuperscript{3}, in the literature the topic of log filtering is not mentioned.

When we consider the available plug-ins none of them targets noise removal. We have for example the Filter Log using Simple Heuristics plug-in which is specialized in the removal of traces which are not starting and/or ending with a specific activity, and in the removal of events belonging to specific activities.

The ProM Framework also offers other types of plug-in. For example, it is possible to filter a log removing all events which do not have a certain attribute, or which the attribute value does not match a given value. Finally, it is also available a timestamp-based filter, which can be used to select events contained in the log occurring during the first six months of recordings.

B. Noise-tolerant Discovery Algorithms

The \textit{\alpha} algorithm \textsuperscript{[?] 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 activities 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 activities: i) causality, represented as \(\rightarrow\), which is discovered if \(a > b\) and \(b \not\sim a\), ii) concurrency, represented as \(|\), which is discovered if \(a > b\) and \(b > a\), and iii) conflict, represented as \(\#\), which is discovered if \(a \not\sim b\) and \(b \not\sim a\). The \(\alpha\) 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 \(\alpha\) algorithm, including its inability to deal with noise, several noise-tolerant discovery algorithms were proposed. The first attempt was the Heuristics Miner [?]. This algorithm discovers a model using the \(\alpha\) relationships. In order to limit the effects of noise, the Heuristics Miner introduces a frequency-based metric \(\alpha\): 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 activities is replaced by the \(\rightarrow\) relationship. A similar approach is also used by Fodina [?], a technique which is based on the Heuristics Miner.

The Inductive Miner [?] 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 (\(\land\)), or iteration (\(\bigcirc\)) - in the graph 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 activities.

The Fuzzy Miner [?], 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 [?] 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” models which suffer from very low precision.

C. Model Dimensions

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

Recall measures how well a model can reproduce the process behavior contained in a log. A recall measurement of 0 indicates the inability to reproduce the behavior recorded in the log while a value of 1 indicates the ability to reproduce all recorded behavior. In order to measure recall we use the approach proposed by Adriansyah et al. [?] 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 recall measurement [?], [?].

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 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 recall, we decided to adopt the approach proposed by Adriansyah et al. [?]. 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 recall and precision in a single measure of model accuracy, and is the harmonic mean of recall and precision \(2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}\).

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 [?]. 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. [?] which we decided not to use since in our tests this approach returns similar results across all discovered models.

\(^{4}\)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.
Finally, complexity quantifies the structural complexity of a process model and can be measured using various complexity metrics [?] such as:

- **Size**: the number of nodes.
- **Control-Flow Complexity (CFC)**: the sum of all connectors (i.e. a place followed by more than two transitions or a place preceded by more than two transitions or a transition followed by more than two places or a transition preceded by more than two places) weighted by their potential combinations of states after a split.
- **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 activities and arcs resulting from the presence of new connections introduced by the noise.

### III. Approach

In this section we present an approach 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.

**A. Preliminaries**

For the purpose of auditing, the execution of processes is generally recorded in an event log. An event log is composed of several **traces**. Each trace is a sequence of events which are associated with a specific task.

**Definition 1** (Event Log). Let \( \Gamma \) be a finite set of tasks. A log \( \mathcal{L} \) is defined as \( \mathcal{L} = (\mathcal{E}, \mathcal{C}, \mathcal{T}, <) \) where \( \mathcal{E} \) is the set of events, \( \mathcal{C} \) is the set of case identifiers, \( \mathcal{T} : \mathcal{E} \rightarrow \mathcal{C} \) is a surjective function linking events to cases, \( T : \mathcal{E} \rightarrow \Gamma \) is a surjective function linking events to tasks, and \( < \subseteq \mathcal{E} \times \mathcal{E} \) 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 \( e_1 < e_2 \land C(e_1) = C(e_2) \land \exists e_3 \in \mathcal{E}[C(e_3) = C(e_1) \land e_3 < e_2 \land C(e_3) \land e_1 < e_3 < e_2]. \)

Given a log, several relations between tasks can be defined based on their underlying events. We are interested in the

---

3 As we are working with only one log the subscript \( L \) will be omitted in the remainder of the paper.

---

**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 \( \exists e_1, e_2 \in \mathcal{E} \land T(e_1) = x \land T(e_2) = y \land e_1 \sqsubset e_2 \).

**B. 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 \( \mathcal{L} \) is defined as a directed graph \( \mathcal{A} = (\Gamma, \rightsquigarrow) \).

For an automaton we can retrieve all initial states through \( \uparrow_{\mathcal{A}} = \{ e \in \Gamma \mid \exists y \in \Gamma[y \rightsquigarrow e] \} \) and all final states through \( \downarrow_{\mathcal{A}} = \{ e \in \Gamma \mid \exists y \in \Gamma[x \rightsquigarrow y] \} \).

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

An arc is considered infrequent iff its relative frequency is a value smaller than a given threshold \( \varepsilon \) 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** (In frequent and Frequent Arcs). The set of infrequent arcs \( \rightsquigarrow^\varepsilon \) is defined as \( \{ (x,y) \in \Gamma \times \Gamma \mid (2 \cdot \#_{\mathcal{A}}((x,y)) / (\#_T(x) + \#_T(y)) < \varepsilon) \land x \rightsquigarrow y \} \). The complement of this set is the set of frequent arcs defined by \( \rightsquigarrow^{m\varepsilon} \triangleq \rightsquigarrow \setminus \rightsquigarrow^\varepsilon \).

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 \( \mathcal{A} \), 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 \( \Gamma^R \) 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. \( \uparrow_{\mathcal{A}} \subset \Gamma^R \) and \( \downarrow_{\mathcal{A}} \subset \Gamma^R \).

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

In order to obtain an anomaly-free automaton \( \mathcal{A}^f \) where the connectivity of required states is not lost, we first consider the set \( \Phi \) which consists of possible arc sets and which is
defined by \( \Phi \triangleq \{ \sigma \in \mathcal{P}(\rightarrow) \mid \sim^m \subset \rightarrow \land \forall s \in \Gamma^R \exists a \in \Gamma^{L \rightarrow} [a \rightarrow s] \land \forall s \in \Gamma^R \exists a \in \Gamma^{L \rightarrow} [s \rightarrow^+ a] \}. \)

We are interested in a (there are potentially multiple candidates) minimal set \( \rightarrow \) in \( \Phi \), i.e. a set from which no more infrequent arcs can be removed. Hence, \( \rightarrow \notin \Phi \) and for all \( V \in \Phi : |V| \geq |\rightarrow| \). The set \( \rightarrow \) is then used to generate our anomaly-free automaton \( \mathcal{A} \triangleq (\Gamma, \Gamma^R, \sim^m, \sim^o \cap \rightarrow) \).

C. 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 approach 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 errors 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. Given a set of events \( E \subseteq \mathcal{E} \) and an anomaly-free automaton \( \mathcal{A} \), this automaton can replay a sequence of two events \( e, e' \in E \), i.e. \( e \rightarrow E e' \), iff \( \exists x, y \in \Gamma [x = T(e) \land y = T(e') \land x \rightarrow y] \). The automaton can replay the entire set of events \( E \), i.e. \( \text{replayable}(E) \), iff \( e_1 \rightarrow E e_2 \rightarrow E e_3 \rightarrow E \ldots \rightarrow E e_n \), where \( E = \{ e_1, e_2, \ldots, e_n \} \), and \( e_1 \) is an event corresponding to an initial state, \( T(e_1) \in \uparrow_{\mathcal{A}} \), and \( e_n \) is an event corresponding to a final state, \( T(e_n) \in \downarrow_{\mathcal{A}} \).

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 the trace corresponding to case \( c \), the set of its subtraces \( \Theta^c \) is defined as \( \Theta^c \triangleq \{ e \in \mathcal{P}(\mathcal{E}^c) \mid \text{replayable}(E) \} \), where \( \mathcal{E}^c \) is the set of events in case \( c \), i.e. \( \{ e \in \mathcal{E} \mid C(e) = c \} \).

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

Definition 8 (Longest Playable Subtrace). Given the trace corresponding to case \( c \), the set of its longest replayable subtraces \( \Theta^c \) is defined as \( \Theta^c \triangleq \{ e \in \mathcal{P}(\mathcal{E}^c) \mid \text{replayable}(E) \} \) such that for all \( \eta \in \Theta^c \) it holds that \( |\Theta^c| \geq |\eta| \).

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

Definition 9 (Filtered Log). The filtered version of \( \mathcal{L} \) is defined as \( \mathcal{F} = (E, \text{ran}(\mathcal{E}^c_i), C^t, T^t, E \times E) \) where \( E \) is defined as \( \bigcup_{\sigma \in \Theta^c} \mathcal{E}^c \).

Figure 2 shows how the approach works. In the example a threshold of \( \varepsilon = 0.05 \) is used. Starting from a log containing infrequent behavior, the log automaton is generated, where \( A \) is the initial state and \( B \) the final state. The frequency of a node \( c \rightarrow \) is reported as a superscript of that node, while the frequency of an arc is reported on the arc itself. It can be observed that in the next version of the automaton two arcs were removed, i.e. \( (B, D) \) and \( (C, B) \). These two arcs are infrequent, e.g. the relative frequency of arc \( (B, D) \) is \( \frac{0.045}{0.05} \). In the subsequent phase this anomaly-free automaton is used to filter the log. In the filtered log, event \( B \) is removed from the last trace since the anomaly-free automaton was not capable of reproducing this event and it was thus treated as infrequent behavior.

IV. The Minimum Log Automaton Problem

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

A. Complexity

The identification of the minimum log 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 [?]!) to the minimum log 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 \( S \subseteq \mathcal{P}(U) \) composed of subsets of \( U \), an instance \( (U, S) \) of the set covering problem consists of identifying the smallest subset of \( S \subseteq U \) such that its union equals \( U \), i.e. \( \bigcup \subseteq S = U \).

Formally, let \( I = (U, S) \) be an instance of a set cover problem, its related log automaton problem is defined as \( \mathcal{A} \triangleq (\Gamma, \Gamma^R, \sim^m, \sim^o) \) where:

\[
\begin{align*}
\Gamma_I & \triangleq \Gamma_{S} \cup \Gamma_U \cup \{ i, o \} \text{ with} \\
\Gamma_{S} & \triangleq \{ s_j \mid j \in S \}; \\
\Gamma_{U} & \triangleq \{ u_k \mid k \in U \}; \\
\Gamma_{I} & \triangleq \Gamma_{S} \cup \{ i, o \}; \\
\sim^m_{I} & \triangleq \{ (s_j, u_k) \mid j \in S \land k \in j \} \cup \Gamma_{U} \times \{ o \}; \\
\sim^o_{I} & \triangleq \{ i \} \times \Gamma_{S}.
\end{align*}
\]

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

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

Lemma 1. Let \( I = (U, S) \) be an instance of the set covering problem that has a solution and let \( \Pi(I) \triangleq (\Gamma, \Gamma^R, \sim^m, \sim^o) \) be its transformation. If \( \mathcal{A}_I \triangleq (\Gamma, \Gamma^R, \sim^m, \sim^o) \) is a minimum log automaton for \( \Pi(I) \) then \( C \triangleq \{ c \in S \mid i \rightarrow c \subseteq_{s_c} \} \) 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 \( u_k \in \mathcal{A}_I \). \( u_k \) is on a path from \( i \) to \( o \). Hence, there is a node \( s_j \) such that \( i \rightarrow s_j \) and \( s_j \rightarrow o \). 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 $\mathcal{A}'' = (\Gamma_i, \Pi_i, \sim_i^m, \sim_i^m)$ with $\sim_i^m = \{ (i, s_c) \in C \}$. Observe that $|\sim_i^m| < |\sim_i^m|$. Let $k \in U$. As $C'$ is a cover, there exists a $j \in C'$ such that $k \in j$. Therefore, $s_j \sim_i^m u_k$. As $i \sim_i^m s_j$ (given that $j \in C'$) and $u_k \sim_i^m o$ (by construction) $u_k$ is on a path from $i$ to $o$ in $\mathcal{A}''$. Hence, $i$, $o$, and all states $u_k$, $k \in U$, are on a path from $i$ to $o$. Therefore in $\mathcal{A}''$ all required states are on a path from $i$ to $o$ and $\mathcal{A}''$ contains fewer infrequent arcs than $\mathcal{A}'$. Hence $\mathcal{A}'$ is not minimal. Contradiction.

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

This follows from the fact that transforming a set covering problem to a minimum log 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 log automaton (see Lemma 1).

**Lemma 2.** Let $I = (U, S)$ be an instance of the set covering problem and let $\Pi(I) \triangleright (\Gamma_i, \Pi_i, \sim_i, \sim_i^m)$ be its transformation. If $C \in S$ is a minimum set cover for $I$ then $\mathcal{A}' = (\Gamma_i, \Pi_i, \sim_i^m, \sim_i^m)$, where $\sim_i^m = \{ (i, s_c) \in C \}$, is a minimum log automaton for $\Pi(I)$.

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

1) All required states are on a path from $i$ to $o$ in $\mathcal{A}'$. Let $k \in U$. As $C$ is a cover, there exists a $j \in C$ such that $k \in j$. Therefore, $s_j \sim_i^m u_k$. As $i \sim_i^m s_j$ (given that $j \in C$) and $u_k \sim_i^m o$ (by construction) $u_k$ is on a path from $i$ to $o$ in $\mathcal{A}'$. Hence, $i$, $o$, and all states $u_k$, $k \in U$, are on a path from $i$ to $o$. Therefore in $\mathcal{A}'$ all required states are on a path from $i$ to $o$.

2) $\mathcal{A}'$ is minimal. Let us assume $\mathcal{A}'$ is not minimal and there exists a minimum log automaton $\mathcal{A}'' = (\Gamma_i, \Pi_i, \sim_i^m, \sim_i^m)$ such that $|\sim_i^m| < |\sim_i^m|$. Define $C = \{ s_c \in S \mid i \sim_i^m s_c \}$. Observe $|C| < |C|$. Since in $\mathcal{A}''$ 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.

![Fig. 2: Example: filtering a log containing infrequent behavior.](image1)

![Fig. 3: Sample Reduction from Set Covering Problem to Minimum Log Automaton Problem.](image2)

**Figure 3** shows how to reduce a set covering problem to a minimum log 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 elements is converted into a state. Additionally, each state is 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 $s1$ is connected to $a, b,$ and $c$. Finally, the initial state is connected to each state representing a subset through a infrequent arc (red arrow).

**B. 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 $n_1 \sim n_2$ 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 $\overline{C}_n \in \{0, 1\}$. If state $n$ reachable from an initial state, the ILP solution assigns to $\overline{C}_n$ a value of 1; otherwise $\overline{C}_n = 0$.
• for each state \( n \in \Gamma \) there exists a variable \( \bar{e}_n \in \{0, 1\} \). If state \( n \) can reach a final state, the ILP solution assigns to \( \bar{e}_n \) a value of 1; otherwise \( \bar{e}_n = 0 \).

• for each arc \( n_1 \rightarrow n_2 \) there exists a variable \( \bar{L}_{n_2,n_1} \in \{0, 1\} \). If state \( n_2 \) is reachable from an initial state through an arc connecting \( n_1 \) to \( n_2 \), the ILP solution assigns to \( \bar{L}_{n_2,n_1} \) a value of 1; otherwise \( \bar{L}_{n_2,n_1} = 0 \).

• for each arc \( n_1 \rightarrow n_2 \) there exists a variable \( \bar{L}_{n_1,n_2} \in \{0, 1\} \). If state \( n_1 \) can reach a final state through an arc connecting \( n_1 \) to \( n_2 \), the ILP solution assigns to \( \bar{L}_{n_1,n_2} \) a value of 1; otherwise \( \bar{L}_{n_1,n_2} = 0 \).

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

\[
\min \sum_{n_1 \in \mathbb{N}} \sum_{n_2 \in \mathbb{N}} e_{n_1,n_2}. \tag{1}
\]

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 \( s \in \uparrow_{\alpha} \), we mark it as reachable from an initial state:
  \[ \bar{e}_s = 1. \tag{3} \]

- For each final state \( t \in \downarrow_{\alpha} \), we mark it as able to reach a final state:
  \[ \bar{e}_t = 1. \tag{4} \]

- For each state \( n \in \Gamma_{R} \), we require it to be reachable from an initial state:
  \[ \bar{e}_n = 1. \tag{5} \]

- For each state \( n \in \Gamma_{R} \), we require it to be able to reach a final state:
  \[ \bar{e}_n = 1. \tag{6} \]

- 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 \( n_2 \), then \( n_2 \) is reachable from an initial state through \( n_1 \):
  \[ L_{n_2,n_1} = 1 \iff \bar{e}_n + e_{n_1,n_2} = 2. \tag{7} \]

- 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 \( n_2 \), then \( n_1 \) is able to reach a final state through \( n_2 \):
  \[ L_{n_1,n_2} = 1 \iff \bar{e}_n + e_{n_1,n_2} = 2. \tag{8} \]

- Each non-initial state \( n_1 \in \Gamma \setminus \downarrow_{\alpha} \) 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:
  \[ \bar{e}_n = 1 \iff \sum_{n_2 \in \mathbb{N}} L_{n_1,n_2} \geq 1. \tag{9} \]

- Each non-final state \( n_1 \in \Gamma \setminus \downarrow_{\alpha} \) 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:
  \[ L_{n_1,n_2} = 1 \iff \sum_{n_2 \in \mathbb{N}} L_{n_1,n_2} \geq 1. \tag{10} \]

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:

\[
2 \cdot L_{n_2,n_1} - C_{n_1} - e_{n_1,n_2} \leq 0, \tag{11}
\]

\[
C_{n_1} + e_{n_1,n_2} - 2 \cdot L_{n_2,n_1} \leq 1. \tag{12}
\]

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

\[
C_{n_1} = x, \quad e_{n_1,n_2} = y.
\]

Now we can rewrite Inequalities 11 as:

\[
2 \cdot x - y \leq 0, \quad y - 2 \cdot x \leq 1.
\]

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 \( y = 2 \). Similarly, the second constraint can only be satisfied if either \( y < 2 \) or \( x = 1 \). 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 equivalence inequalities of the form shown below:

\[
C_{n_1} - \sum_{n_2 \in \mathbb{N}} L_{n_1,n_2} \leq 0, \tag{12}
\]

\[
\sum_{n_2 \in \mathbb{N}} L_{n_1,n_2} - M \cdot C_{n_1} \leq 0.
\]

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

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

\[
C_{n_1} = x, \quad \sum_{n_2 \in \mathbb{N}} L_{n_1,n_2} = y.
\]

Now we can rewrite Inequalities 12 as:

\[
x - y \leq 0, \quad y - M \cdot x \leq 0.
\]

Considering that \( x \) can only be 0 or 1, and \( y \geq 0 \), the first constraint can only be satisfied if either \( x = 0 \) 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. □
V. Frequency Threshold Identification

In section III-B we introduced the concepts of frequent and infrequent arcs in a log automaton. These concepts are based on a frequency threshold $\varepsilon$. In this section we present an approach for the automated identification of such a threshold.

The idea behind the approach 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. $IQR = Q_3 - Q_1$. In particular the lower-half interquartile range is defined as the difference between the median and the lower quartile, $IQR_L = median - Q_1$. 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, $\varepsilon$ can be defined as a value $x \in [0, \Lambda_\lambda (\neg \neg \neg \neg y)]$ for which:

$$\rho_{IQR}(\neg \neg \neg \neg m_x) \leq 1 \land \forall y \in [0, 1] [\rho_{IQR}(\neg \neg \neg \neg y) \leq 1 \land |\neg \neg \neg \neg y| > |\neg \neg \neg \neg m_y|],$$

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

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.

VI. 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\footnote{Available at https://svn.win.tue.nl/trac/prom/browser/Packages/NoiseFiltering}. The plug-in can use either Gurobi\footnote{http://www.gurobi.com} or LPsolve\footnote{http://lpsolve.sourceforge.net} as ILP solver.

To identify infrequent events we used the alignment-based replay technique proposed in [?]. 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 sync 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, a fictitious 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 [?]. A Workflow net is easy sound if there is at least one firing sequence that can reach the final marking [?]. 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).

A. Design

The design for the three experiments is illustrated in Figure 4. 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 copes infrequent behavior in a controlled environment. The first 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 recall and precision of the technique.

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 II.

Finally, in the third experiment, we repeated the same procedure using various real-life logs. 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 [?], Heuristics
Miner [?], Fodina [?] and ILP Miner [?]. 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.

We set the \( \lambda \) percentile to 0.125, 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 ILP solver.

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

B. Datasets

For the first two experiments, we generated a base log using CPN Tools\(^{10}\) and from this base log we produced 8 “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). For the second experiment we used the same set of logs. We generated the infrequent behavior by inserting new events in the log. The labels of these events were selected in such a way that the insertion of a new 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 noisy events were to be inserted.

For the third experiment, we used four real-life logs from different domains and of different size in order to be able to generalize the evaluation results. Specifically, we used logs from financial and medical institutions, and from Australian and Dutch companies. Two such logs are publicly available from financial and medical institutions, and from Australian generalize the evaluation results. Specifically, we used logs different domains and of different size in order to be able to insert dependency that was already present in the original log. The labels of these events were selected in such a way that the insertion of a new 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 noisy events were to be inserted.

For the third experiment, we used four real-life logs from different domains and of different size in order to be able to generalize the evaluation results. Specifically, we used logs from financial and medical institutions, and from Australian and Dutch companies. Two such logs are publicly available from financial and medical institutions, and from Australian generalize the evaluation results. Specifically, we used logs different domains and of different size in order to be able to insert dependency that was already present in the original log. The labels of these events were selected in such a way that the insertion of a new 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 noisy events were to be inserted.

C. Results

As for the first experiment, Figure 5 plots the recall and precision of our technique at different levels of infrequent behavior. The graph shows the recall of the algorithm to be above 90% in all logs independently of the level of infrequent behavior while the precision never drops below 0.68.

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 approach 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 6 shows the results of recall, precision, F-score, model size and generalization dimensions, obtained through application of the baseline discovery algorithms, with and without our technique, on the

Table I 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%.

<table>
<thead>
<tr>
<th>Artificial Log</th>
<th>#Traces</th>
<th>#Events</th>
<th>#Unique Labels</th>
<th>%Infrequent Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>N10</td>
<td>2239</td>
<td>13359</td>
<td>13</td>
<td>5%</td>
</tr>
<tr>
<td>N15</td>
<td>2239</td>
<td>15714</td>
<td>13</td>
<td>15%</td>
</tr>
<tr>
<td>N20</td>
<td>2239</td>
<td>16101</td>
<td>13</td>
<td>20%</td>
</tr>
<tr>
<td>N25</td>
<td>2239</td>
<td>17175</td>
<td>13</td>
<td>25%</td>
</tr>
<tr>
<td>N30</td>
<td>2239</td>
<td>18401</td>
<td>13</td>
<td>30%</td>
</tr>
<tr>
<td>N35</td>
<td>2239</td>
<td>19817</td>
<td>13</td>
<td>35%</td>
</tr>
<tr>
<td>N40</td>
<td>2239</td>
<td>21468</td>
<td>13</td>
<td>40%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Real-Life Log</th>
<th>#Traces</th>
<th>#Events</th>
<th>#Unique Labels</th>
<th>%Infrequent Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP2012</td>
<td>13087</td>
<td>148192</td>
<td>15</td>
<td>39%</td>
</tr>
<tr>
<td>BP2014</td>
<td>46616</td>
<td>422563</td>
<td>9</td>
<td>13%</td>
</tr>
<tr>
<td>Hospital11</td>
<td>688</td>
<td>9575</td>
<td>19</td>
<td>2%</td>
</tr>
<tr>
<td>Hospital2</td>
<td>617</td>
<td>9666</td>
<td>22</td>
<td>2%</td>
</tr>
</tbody>
</table>

TABLE I: Characteristics of the logs used in the evaluation.
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 0.863 instead of 0.320, with Mann-Whitney test U = 903, z = 5.268, p = 0.000 < 0.05). This significant increment is explained by the noticeable and significant increment of precision (Filtered Mdn 0.762 instead of 0.194, with Mann-Whitney test U = 896, z = 5.174, p = 0.000 < 0.05) and less noticeable and significant increment of recall (Mdn 0.995 instead of 0.922, with Mann-Whitney test U = 682.5, z = 2.336, p = 0.019 < 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 recall 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 54 after the application (Mann-Whitney test U = 203, z = 4.158, p = 0.000 < 0.05). Table II 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 6. The increase in density is expected, as this metric is inversely correlated with size (smaller models tend to be denser) [?].

Finally, our technique improves recall, precision, and complexity without negatively affecting generalization. In fact, the latter remains constant to a median of 0.997 (Mann-Whitney test: U = 512, z = 512, p = 1 > 0.05).

![Fig. 5: Recall and precision of the technique at different levels of infrequent behavior.](image)

![Fig. 6: Recall, Precision, F-Score, Size and Generalization comparison between filtered and original log using different artificial logs and discovery algorithms.](image)
<table>
<thead>
<tr>
<th>Log</th>
<th>Measure</th>
<th>Inductive</th>
<th>Heuristics</th>
<th>Fodina</th>
<th>ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Original</td>
<td>Filtered</td>
<td>Original</td>
<td>Filtered</td>
</tr>
<tr>
<td>N5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNC</td>
<td>3.818</td>
<td>1.209</td>
<td>1.422</td>
<td>1.167</td>
<td>1.338</td>
</tr>
<tr>
<td>Density</td>
<td>0.019</td>
<td>0.029</td>
<td>0.017</td>
<td>0.016</td>
<td>0.021</td>
</tr>
<tr>
<td>N10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNC</td>
<td>3.818</td>
<td>1.209</td>
<td>2.949</td>
<td>1.167</td>
<td>1.506</td>
</tr>
<tr>
<td>Density</td>
<td>0.019</td>
<td>0.029</td>
<td>0.022</td>
<td>0.016</td>
<td>0.019</td>
</tr>
<tr>
<td>N15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNC</td>
<td>3.818</td>
<td>1.209</td>
<td>3.320</td>
<td>1.167</td>
<td>1.500</td>
</tr>
<tr>
<td>Density</td>
<td>0.024</td>
<td>0.029</td>
<td>0.022</td>
<td>0.016</td>
<td>0.019</td>
</tr>
<tr>
<td>N20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>0.022</td>
<td>0.029</td>
<td>0.020</td>
<td>0.016</td>
<td>0.018</td>
</tr>
<tr>
<td>N25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNC</td>
<td>3.818</td>
<td>1.209</td>
<td>6.967</td>
<td>1.167</td>
<td>1.849</td>
</tr>
<tr>
<td>Density</td>
<td>0.022</td>
<td>0.029</td>
<td>0.020</td>
<td>0.016</td>
<td>0.018</td>
</tr>
<tr>
<td>N30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNC</td>
<td>3.818</td>
<td>1.209</td>
<td>2.654</td>
<td>1.167</td>
<td>1.849</td>
</tr>
<tr>
<td>Density</td>
<td>0.022</td>
<td>0.029</td>
<td>0.020</td>
<td>0.016</td>
<td>0.018</td>
</tr>
<tr>
<td>N35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>0.022</td>
<td>0.029</td>
<td>0.020</td>
<td>0.016</td>
<td>0.020</td>
</tr>
<tr>
<td>N40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>0.022</td>
<td>0.029</td>
<td>0.020</td>
<td>0.016</td>
<td>0.020</td>
</tr>
<tr>
<td>BPI 2012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>0.022</td>
<td>0.029</td>
<td>0.020</td>
<td>0.016</td>
<td>0.020</td>
</tr>
<tr>
<td>BPI 2014</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>0.022</td>
<td>0.029</td>
<td>0.020</td>
<td>0.016</td>
<td>0.020</td>
</tr>
<tr>
<td>Hospital 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>0.022</td>
<td>0.029</td>
<td>0.020</td>
<td>0.016</td>
<td>0.020</td>
</tr>
<tr>
<td>Hospital 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>0.022</td>
<td>0.029</td>
<td>0.020</td>
<td>0.016</td>
<td>0.020</td>
</tr>
</tbody>
</table>

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

The results on real-life logs, summarized in Figure 7, are in line with those obtained on artificial logs. The F-score significantly improves (Mdn 0.681 instead of 0.524, with Mann-Whitney test U = 194, z = 2.488, p = 0.012 < 0.05) due to a significant improvement in precision (Mdn 0.573 instead of 0.368, with Mann-Whitney test U = 197.5, z = 2.620, p = 0.007 < 0.05) despite a reduction in recall (Mdn 0.855 instead of 0.905, with Mann-Whitney test U = 67, z = -2.299, p = 0.021 < 0.05). The size of the discovered model is again significantly reduced from a median of 91 elements to a median of 58 elements (Mann-Whitney test U = 53, z = -2.827, p = 0.004 < 0.05). Similarly, in Table II we can see that CNC, ACD, and Fodina decrease also for the real-life logs. Finally, generalization slightly decreases from a median of 0.892 to a median of 0.871, which is not significant (Mann-Whitney test U = 104, z = -0.905, p = 0.381 > 0.05).

Time performance. In the experiment with real-life logs, the technique took on average 150 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 III. As we can see, time performance is within reasonable bounds.
VII. Conclusion

In this paper we presented a technique for the automatic removal of infrequent behavior from process execution logs. The core idea of this technique 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 through the removal of individual events using alignment-based replay [7].

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 statistically significant improvement over fitness, precision and complexity without a statistically significant negative effect on generalization. Furthermore, the technique generally completes within seconds, with the worst reported case being 14 minutes for a large log. In summary, our filtering technique provides clear advantages over manual data cleaning – a challenging and time consuming task in process mining [7].

By relying on alignment, the technique guarantees that the number of events removed from the original log is minimal, given a set of infrequent event dependencies. Moreover, using ILP we guarantee that the set of such dependencies is maximal. In future, we plan to consider other types of event dependencies, e.g. transitive ones. Another avenue for future work is to develop an approach to isolate behavior (frequent or infrequent) in order to compare logs with similar behaviors.

Acknowledgments. NICTA is funded by the Australian Government via the Department of Communications. This research is funded by the ARC Discovery Project DP150103356, and supported by the Australian Centre for Health Services Innovation SG00009-000450.