Noise Filtering of Process Execution Logs based on Outliers Detection

Raffaele Conforti1, Marcello La Rosa1,2, and Arthur H.M. ter Hofstede1,3

1 Queensland University of Technology, Australia
   {raffaele.conforti, m.larosa, a.terhofstede}@qut.edu.au
2 NICTA Queensland Lab, Australia
3 Eindhoven University of Technology, the Netherlands

Abstract. This paper presents a technique for the automated removal of noise from process execution logs. Noise is the result of data quality issues such as logging errors and manifests itself in the form of infrequent process behavior. The proposed technique generates an abstract representation of an event log as an automaton capturing the direct follows relations between event labels. This automaton is then pruned from arcs with low relative frequency and used to remove from the log those events not fitting the automaton, which are identified as outliers. The technique has been extensively evaluated on top of various automated process discovery algorithms using both artificial logs with different levels of noise, as well as a variety of real-life logs. The results show that the technique significantly improves the quality of the discovered process model along fitness, appropriateness and simplicity, without negative effects on generalization. Further, the technique scales well to large and complex logs.

1 Introduction

Process mining aims to extract actionable process knowledge from the event logs of IT systems that are commonly available within organizations [18]. One way to achieve this goal is by representing the process behavior captured in the log via a process model. Over time, a number of algorithms for automated process model discovery have been proposed, which strike different trade offs between the accuracy in capturing the process behavior recorded in the log, and the complexity of the produced model [18].

The starting assumption of automated process model discovery is that an event log is a faithful snapshot of the process behavior executed in a given period of time within an organization. However, real-life process execution logs, like any other type of event log, often contain “noise” [16, 17] due to data quality issues (e.g. data entry errors or incompleteness). This noise generally leads to new process behavior in the form of infrequent order dependencies between events, which affect the “reliability” of the log as a proxy for process behavior. For example, a gap in the log may lead to new order dependencies between the events before and after the gap, which do not exist in reality.

The inability to effectively detect and filter out noise has a negative effect on the accuracy and simplicity of the model discovered. In fact, the tests reported in this paper show that low levels of noise already have a detrimental effect on the quality of the models produced by various discovery algorithms such as Heuristics Miner [23],
Fodina [22], and Inductive Miner [11], despite these algorithms claim noise-tolerant capabilities. For example the Heuristics Miner, which employs a technique for disambiguating event dependencies due to noise, can have a 45% drop in accuracy when the level of noise is 3% of the total log size.

In this paper, we propose a technique for systematically filtering noise from process execution logs taking inspiration from outliers detection in statistics [14]. The technique builds an abstraction of the event log in the form of an automaton which only captures the direct follow dependencies between the labels of the events in the log. From this automaton we remove those edges (i.e. those dependencies between labels) that are statistically infrequent and are thus regarded as outliers. Next, we replay the original log onto the modified automaton in order to identify unfitting events, and remove these events from the log. The technique aims to maximize the removal of outlier dependencies from the automaton and minimizes the number of events to be removed from the log in order to guarantee a filtered log that perfectly fits the automaton.

Fig. 1: BPMN model obtained from InductiveMiner on a hospital log before and after applying the proposed technique.²

The technique has been implemented on top of the ProM framework and extensively evaluated in combination with different baseline discovery algorithms, using a two-pronged approach. First, we evaluated the improvement of accuracy and complexity over the baseline algorithms in the presence of varying levels of noise, using artificially-generated logs, so as to control the level of noise. We then repeated the tests using a variety of real-life logs exhibiting different characteristics in terms of size and number of (distinct) events. We measured accuracy using the well-established measures of fitness (i.e. recall) and appropriateness (i.e. precision), while we used different

² Different labels between the two models are due to the events filtering of InductiveMiner.
structural complexity measures such as size, density and control-flow complexity, as proxies for model complexity. The results show a statistically significant improvement of fitness, appropriateness and complexity when using the proposed technique, without affecting the generalization of the underlying automated discovery algorithm. As an example, Fig. 1 shows the BPMN model obtained by Inductive Miner on the log of an Australian hospital, before and after the application of the technique.\(^3\) Time performance tests show that the technique scales well to large and complex logs, being able to preprocess a log generally in a matter of seconds.

The paper is structured as follows. Section 2 discussed algorithms for automated process discovery with a focus on noise tolerance, and how the topic of noise filtering has been addressed in statistics and data mining. Next, Section 3 illustrates the proposed technique while Section 4 discusses the results of its evaluation. Finally, Section 5 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. Finally, we review approaches for outliers detection in the fields of statistics and data mining.

2.1 Noise Tolerant Discovery Algorithms

The \(\alpha\) algorithm 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 between two activities one of the following relations exists: causality, indicated as \(\rightarrow\) and discovered if \(a > b\) and \(b \nRightarrow a\), concurrency, indicated as \(\parallel\) relationship and discovered if \(a > b\) and \(b > a\), and conflict, indicated as \(#\) and discovered if \(a \nRightarrow b\) and \(b \nRightarrow a\). The \(\alpha\) algorithm assumes that a log is complete and free from noise, and produces unsound models if this is not the case.

In order to overtake the limitations of the \(\alpha\) algorithm and due to the absence of noise filtering techniques, several noise-tolerant discovery algorithms were proposed. The first attempt was the Heuristics Miner [23]. 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 \(\Rightarrow\): 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 \(\parallel\) relationship has been correctly identified and if the value of \(\Rightarrow\) is above a given threshold, a \(\parallel\) relationship will be replaced by a \(\rightarrow\) relationship. A similar approach is also used by Fodina [22] which is based on the Heuristics Miner.

The Inductive Miner [11] is a discovery algorithm based on a divide-and-conquer approach. This algorithm, using the direct follows dependency, generates the directly-follows graph. Next, it identifies a cut (i.e. \(\times\), \(\rightarrow\), \(\wedge\), and \(\bigcirc\)) in the graph along which the log is split. This operation is repeated recursively until no more cuts can be identified.

\(^3\) Activity labels have been masked to preserve anonymity of the hospital.
The mining is then performed on the portions of the log discovered using the cuts. In order to deal with noise, the algorithm applies a couple of filters. The first is a filter in the style of the Heuristics Miner, which removes edges from the directly-follows graph. In addition, it uses the \textit{eventually-follows graph} to remove edges which the first filter did not removed.

These approaches for handling noise present two limitations. First, dependencies are removed only if they are “ambiguous”, e.g. replacing a $\parallel$ dependency with a $\rightarrow$ dependency, does not remove dependencies which are simply infrequent. Second, dependencies removed as part of the filtering are only removed from the dependency graph, and not from the log, influencing the final result of the discovery.

The \textit{Fuzzy Miner} [6], 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 \textit{fuzzy net} where each node and edge is associated with a value of correlation and significance. After the mining phase, the user 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). 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 \textit{ILP miner} [21] 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” models which suffer from very low precision.

\subsection{Model Dimensions}

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

\textit{Recall} measures how well a model can reproduce the process behavior present in the log. A recall measurement with a value of 0 indicates the inability to reproduce the behavior recorded in the log while a value of 1 indicates the ability to reproduce all the behavior. In order to measure recall we use the approach proposed by Adriansyah et al. [2] 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 [5, 11].

\textit{Precision} measures the capability of a model to reproduce only the behavior recorded in the log. A value of 0 indicates that the model can reproduce behavior never observed in the log while a value of 1 indicates that the model only reproduces the 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. [1]. Accordingly, after generating an alignment automaton describing the set of exe-
cuted 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 \( \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}} \).

Generalization can be seen as the opposite of precision. It provides a measurement over the capability of the model to reproduce behavior never observed in the log. We decided to measure generalization using 10-fold cross validation, which is an established approach in data mining [9]). Accordingly, a log is divided into 10 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. [19] which we decided not to use since in our tests this approach returns a very similar result across all discovered models.

Finally, complexity quantifies the structural complexity of the process model and can be measured using various complexity metrics [12] such as:

- Size: the number of nodes.
- Control-Flow Complexity (CFC): the sum of all connectors (i.e. a place/transition followed/preceded by more than two transitions/places) weighted by their potential combinations of states after a split.
- Average Connector Degree (ACD): the average number of nodes a connector (i.e. a place/transition followed/preceded by more than two transitions/places) 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 any model with the same number of nodes.

Noise affects the above quality dimensions in different ways. Recall is not reliable since it is be computed on a log containing noise, i.e. a high value of recall does not necessarily guarantee that the discovered model is an exact representation of reality. Precision tends to be lower since noise in the log introduces new connections between event labels which should not be there. Generalization on the other side tends to be higher due to the increased number of connections. Finally, complexity tends be higher since the model is more complex as it contains more activities and more arcs.

2.3 Outliers Detection

The concept of noise can be compared to the concept of outlier in statistics and data mining. In these fields outliers detection is the labeling and identification of observations which deviate too much from the other observations.

When we look at approaches for outliers detection we can categorize them in two groups: global and local outlier models [10]. Global outlier models are based on statistical models. We can find three types of approaches: statistical tests, depth-based approaches and deviation-based approaches. Statistical tests [14] assume the data to be normally distributed, and unlikely observations (based on mean and standard deviation) are considered as outliers. Depth-based approaches [7, 15] follow a similar idea but no assumption about the distribution is made. In this case, data is used to discover a convex hull which has normal observations in its centre and unlikely observations on its
borders. Deviation-based approaches [3] are based on the idea that normal observations share similar characteristics and the removal of outliers contributes to a minimization of the variance in the set of observations.

Local outlier models are based on spatial proximity. We can distinguish between distance-based and density-based approaches. Distance-based approaches [8] assume that normal observations have a dense neighborhood while outliers are located far away from their neighbors. Density-based approaches [4] assume that the density around a normal observation is similar to the density in the neighborhood while the density around an outlier differs considerably from the density in the neighborhood.

These approaches are not applicable to our problem. Local outlier models cannot be used since they rely on the concept of spatial proximity, and in the context of a process execution log, where a trace is a sequence of events, spatial proximity between events cannot be easily defined. Global outlier models, on the other hand, are not applicable since they would consider events belonging to infrequent activities as outliers.

3 Approach

In this section we present an approach for noise filtering based on outliers detection. After introducing preliminary concepts such as event log and direct follow dependencies, the concept of log automaton is presented. The identification of outliers in a log automaton and their use for noise removal concludes the section.

3.1 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 [20]). Let $\Gamma$ be a finite set of tasks. A log $L$ is defined as $L = (\mathcal{E}, \mathcal{C}, T, <)$ where $\mathcal{E}$ is the set of events, $\mathcal{C}$ is the set of trace identifiers, $C : \mathcal{E} \rightarrow \mathcal{C}$ is a surjective function linking events to traces, $T : \mathcal{E} \rightarrow \Gamma$ 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 $e_1 < e_2 \land C(e_1) = C(e_2) \land \lnot \exists e_3 \in \mathcal{E} [C(e_3) = C(e_1) \land e_1 < e_3 \land e_3 < e_2].$

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

As we are working with only one log the subscript $L$ will be omitted in the remainder of the paper.
3.2 Noise detection

In this section we present a technique for noise detection which relies on the identification of outliers in a so-called log automaton. In this context, outliers represent relations, which are captured through arcs, 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 \( L \) is defined as a directed graph \( A = (\Gamma, \Rightarrow) \).

For an automaton we can retrieve all initial states through \( ↑A = \{ x ∈ \Gamma \mid \not\exists y ∈ \Gamma[ y ⇓ x ] \} \) and all final states through \( ↓A = \{ x ∈ \Gamma \mid \not\exists y ∈ \Gamma[ x ⇓ y ] \} \).

As we are interested in frequencies of task occurrences and of direct follow dependencies, we introduce the function \( \#Γ : \Gamma → \mathbb{N} \) defined by \( \#Γ(x) = |\{ z ∈ \mathcal{E} | T(z) = x \}| \) and the function \( \#\Rightarrow : \Rightarrow → \mathbb{N} \) defined by \( \#\Rightarrow(x, y) = |\{(e₁, e₂) ∈ \mathcal{E} × \mathcal{E} | T(e₁) = x ∧ T(e₂) = y ∧ e₁ ⊏ e₂ \}| \).

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 \( \Delta \) is defined as \( \{(x, y) ∈ \Gamma × \Gamma \mid (\#\Rightarrow(x, y))/(\#Γ(x) + \#Γ(y)) < ε \} \). The complement of this set is the set of frequent arcs defined by \( Π ≜ \Rightarrow \setminus \Delta \).

The indiscriminate removal of outliers from a log automaton may result in an automaton where certain states can no longer be reached from an initial state or from which final states can no longer be reached. In order to obtain an outlier free automaton \( Λ \) where this connectivity is not lost, first we consider the set \( Φ \) which consists of possible arc sets and which is defined by \( Φ ≜ \{ \neg → ∈ \mathcal{P}(\Rightarrow) \mid Π \subseteq → ∧ \forall s ∈ \Gamma∃ a ∈ ↑(Γ, →) [a → + s] ∧ \forall s ∈ \Gamma∃ a ∈ ↓(Γ, →) [s → + a] \} \). We are interested in a (there are potentially multiple candidates) minimal set \( → \) in \( Φ \), i.e. a set from which no more infrequent arcs can be removed. Hence, \( → ∈ Φ \) and for all \( V ∈ Φ | V | ≥ | → | \). The set \( → \) is then used to generate our outlier free automaton \( Λ ≜ (Γ, →) \).

3.3 Noise removal

In this section focus is on noise removal in an automaton where the outliers have been removed as described in the previous section.

The idea behind our approach is inspired by the observation that noise 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 noise removal is to focus on incorrectly recorded events. To this end, events that cannot be replayed on the outlier free automaton are removed.
Definition 5. Given a set of events \( E \subseteq \mathcal{E} \) and an outlier free automaton \( \Lambda \), this automaton can replay a sequence of two events \( e_1, e_2 \in E \), i.e. \( e_1 \rightarrow e_2 \), iff \( \exists x, y \in \Gamma \mid x = T(e_1) \wedge y = T(e_2) \wedge x \rightarrow y \). The automaton can replay the entire set of events \( \mathcal{E} \), i.e. \( \text{replayable}(E) \), iff there exist \( e_1, e_n \in E \) such that \( e_1 \rightarrow_\mathcal{E} e_n \) and there are no events \( e_0, e_{n+1} \in \mathcal{E} \) such that \( e_0 < e_1 \) and \( e_n < e_{n+1} \).

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 6 (Subtrace). Given a trace in case \( c \), the set of its subtraces \( \Theta^c \) is defined as \( \Theta^c \equiv \{ E \in \mathcal{P}(\mathcal{E}) \mid C(e) = c \} \mid \text{replayable}(E) \} \).

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

Definition 7 (Longest Replayable Subtrace). Given a trace in case \( c \), the set of its longest replayable subtraces \( \Theta^c \) is defined as \( \Theta^c \in \Theta^c \) such that for all \( \eta \in \Theta^c \) it is that case that \( |\Theta^c| \geq |\eta| \).

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

Definition 8 (Filtered Log). The filtered version of log \( \mathcal{L} \), \( \mathcal{F} \), is defined as \( (E, \mathcal{E}'_\mathcal{E}, C|_\mathcal{E}, T|_\mathcal{E}, <\mathcal{E} \times \mathcal{E}) \) where \( E \) is defined as \( \bigcup_{\mathcal{E} \subseteq E} \theta^c \).

Figure 2 shows how the approach works. In the example a threshold of \( \varepsilon = 0.05 \) is used. Starting from a log containing noise, the log automaton is generated. The frequency of a node 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 \( \frac{1}{175} = 0.045 \) < 0.05. In the subsequent phase this outlier free automaton is used to filter the log. In the filtered log, event \( B \) is removed from the last trace since the outlier free automaton was not capable of reproducing this event and it was thus treated as noise.

![Fig. 2: Example: filtering a log containing noise.](image-url)
4 Evaluation

In this section we present the results of two experiments to assess the goodness of our filtering technique. To perform these experiments, we implemented the technique as a plugin, namely the “Noise Filtering” plugin, for the ProM framework.\footnote{Available at https://svn.win.tue.nl/trac/prom/browser/Packages/NoiseFiltering}

To identify the best outlier-free automaton we used a random search algorithm\cite{13} while to identify noisy events we used replay-based alignment\cite{2}. Specifically, we convert the automaton into a Petri net with a single source and a single sink. We then perform the alignment between the Petri net and the log, so as to remove from the log all events identified by a move on log only, i.e. those events that exist in the log but cannot be observed in the automaton. The alignment is repeated until no more such events can be found. We decided to use the replay-based alignment since it guarantees optimality under the assumption that the Petri net is easy sound.

4.1 Design

The first experiment aimed at measuring how our technique copes with noise in a controlled environment. For this we used artificially generated logs where we incrementally injected noise. The second experiment, performed on real-life logs, aimed at verifying if the same levels of performance can be achieved using real-life logs.

In the first experiment starting from an artificial log, we generated several logs by injecting different degrees of noise. These logs were provided as input to several baseline discovery algorithms, before and after applying our noise filtering technique. Finally, we measured the quality of the discovered models against the original log using the metrics described in Section 2.

In the second experiment, we repeated the same procedure using various real-life logs. The experiments design are illustrated in Fig. 3.

![Fig. 3: Experiments setup for artificial and real-life logs.](image)

We used the following discovery algorithms: InductiveMiner\cite{11}, Heuristics Miner\cite{23}, Fodina\cite{22} and ILP Miner\cite{21}. 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 mining result and not on its absolute value.

We set the outliers detection threshold to 0.05, which is a well-accepted value in statistics, and used this to filter out infrequent order dependencies from the automaton.
The results of this evaluation, as well as the artificial datasets that we generated, are provided with the software distribution.

4.2 Datasets

For the first experiment, we generated a base log using CPN Tools and then from this we produced six “noisy” logs by injecting an incremental amount of noise into the base log, as a percentage of its total number of events. We used the following percentages: 5%, 10%, 15%, 20%, 25%, and 30% in order to simulate various levels of noise in real-life logs. We generated the noise by inserting new events in the log. We selected the label of each such event using the uniform distribution from the labels present in the log (i.e. no new label was inserted). We also used a uniform distribution to select trace and position within the trace where the noisy event is inserted.

For the second experiment, we used six real-life logs which exhibit different characteristics in size and domain 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 and are those used for 2012\(^7\) and 2014\(^8\) editions of the BPI Challenge. These two logs were pre-filtered removing infrequent labels (using the Filter Log using Simple Heuristics plugin of ProM with a threshold of 90%).

<table>
<thead>
<tr>
<th>Artificial Log</th>
<th>#Traces</th>
<th>#Events</th>
<th>#Unique Labels</th>
<th>%Noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>N5</td>
<td>3000</td>
<td>34627</td>
<td>13</td>
<td>5%</td>
</tr>
<tr>
<td>N10</td>
<td>3000</td>
<td>36551</td>
<td>13</td>
<td>10%</td>
</tr>
<tr>
<td>N15</td>
<td>3000</td>
<td>38701</td>
<td>13</td>
<td>15%</td>
</tr>
<tr>
<td>N20</td>
<td>3000</td>
<td>41120</td>
<td>13</td>
<td>20%</td>
</tr>
<tr>
<td>N25</td>
<td>3000</td>
<td>43861</td>
<td>13</td>
<td>25%</td>
</tr>
<tr>
<td>N30</td>
<td>3000</td>
<td>46994</td>
<td>13</td>
<td>30%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Real-life Log</th>
<th>#Traces</th>
<th>#Events</th>
<th>#Unique Labels</th>
<th>%Noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPI2012</td>
<td>13087</td>
<td>148192</td>
<td>15</td>
<td>25%</td>
</tr>
<tr>
<td>Insurance1</td>
<td>37335</td>
<td>163224</td>
<td>20</td>
<td>25%</td>
</tr>
<tr>
<td>Insurance2</td>
<td>896</td>
<td>12437</td>
<td>9</td>
<td>35%</td>
</tr>
<tr>
<td>BPI2014</td>
<td>46616</td>
<td>422563</td>
<td>9</td>
<td>37%</td>
</tr>
<tr>
<td>Hospital1</td>
<td>688</td>
<td>9275</td>
<td>19</td>
<td>79%</td>
</tr>
<tr>
<td>Hospital2</td>
<td>617</td>
<td>9666</td>
<td>22</td>
<td>80%</td>
</tr>
</tbody>
</table>

Table 1: Characteristics of the logs used in the evaluation.

Table 1 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 noise. The latter is the percentage of noisy events added for the artificial logs, and the percentage of noisy events removed from the real-life logs, given that for the latter we did not have a noise-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

\(^7\) doi:10.4121/uuid:3926db30-f712-4394-aecb-75976070e91f
\(^8\) doi:10.4121/uuid:86977bac-f874-49cf-8337-80f26bf5d2ef
level of noise observed varies significantly, from 5% to 80%. Interestingly, we observe that the real-life logs used exhibit much higher levels of noise than the artificial logs.

### 4.3 Results

Figure 4 shows the results along F-score, model size and generalization, obtained by the baseline discovery algorithms before and after using our technique on the artificial logs. Missing values such as the F-score on N5 before filtering with Heuristics Miner, are due to the inability of this algorithm to guarantee the soundness of the discovered model [11] – a necessary condition for computing recall and precision using alignment.

From these results we can draw a number of observations. First, Heuristics Miner and Fodina, do present a 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 are able to 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 0.2 for ILP Miner).

Second, and most importantly, the results confirm the effectiveness of our technique. The F-score significantly improves compared to when the technique is not used (Mdn 0.631 instead of 0.224, with Mann-Whitney test: U = 57, z = 4.763, p = 0.000 < 0.01). This significant increment is explained by the significant increment of precision (Mdn 0.521 instead of 0.131, with Mann-Whitney test U = 57, z = 4.764, p = 0.000 < 0.01). Such increment of F-score is less noticeable on models generated by the ILP Miner. This is because the ILP miner is prone to generate “flower” models which suffer from low precision, in order to fit every trace into the model.

Third, our filtering technique also reduces the complexity of the discovered models in a statistically significant way. Before the application of our technique, the discovered model has a median of 156 nodes, which is reduced to 60.5 after using our technique (Mann-Whitney test: U = 459.5, z = 3.54, p = 0.000 < 0.01). Table 2 reports the measurements of the other structural complexity metrics: the decrease in CFC, ACD, CNC confirm the results on size. We observe that the increase in density is expected, as this metric is inversely correlated with size (smaller models tend to be denser) [12].

Fourth, our technique is able to improve recall, precision and complexity without negatively affecting generalization. In fact, the latter only drops from a median of 0.742 to a median of 0.705 (Mann-Whitney test: U = 326.0, z = 0.784, p = 0.433 > 0.01).

The results on real-life logs, summarized in Fig. 5, are in line with those obtained on artificial logs. The F-score significantly improves (Mdn of 0.775 instead of 0.563 with Mann-Whitney test: U = 153.0, z = -2.784, p = 0.005 < 0.01) due to a significant improvement in precision (Mdn of 0.746 instead of 0.401 with Mann-Whitney test: U = 138.5, z = -3.083, p = 0.002 < 0.01). The size of the discovered model is again significantly reduced from a median of 83 elements to a median of 47 elements (Mann-Whitney test: U = 454.0 z = 3.424, p = 0.001 < 0.01). Similarly, in Table 2 we can see that CFC, ACD, CNC decrease also for the real-life logs. Finally, generalization slightly decreases from a median of 0.914 to a median of 0.860, despite not significantly
Fig. 4: F-Score, size and generalization comparison between filtered and original log using different artificial logs and discovery algorithms.
Fig. 5: F-Score, size and generalization comparison between filtered and original log using different real-life logs and discovery algorithms.
<table>
<thead>
<tr>
<th>Log</th>
<th>Measure</th>
<th>Inductive</th>
<th>Heuristics</th>
<th>Fodina</th>
<th>IL.P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Filtered</td>
<td>Original</td>
<td>Filtered</td>
<td>Original</td>
</tr>
<tr>
<td>N5</td>
<td>CNC</td>
<td>3.133</td>
<td>3.269</td>
<td>2.410</td>
<td>1.359</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.021</td>
<td>0.015</td>
<td>0.014</td>
<td>0.018</td>
</tr>
<tr>
<td>N10</td>
<td>CNC</td>
<td>3.313</td>
<td>3.209</td>
<td>2.410</td>
<td>1.359</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.021</td>
<td>0.015</td>
<td>0.014</td>
<td>0.018</td>
</tr>
<tr>
<td>N15</td>
<td>CNC</td>
<td>3.823</td>
<td>1.244</td>
<td>3.378</td>
<td>1.324</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.021</td>
<td>0.016</td>
<td>0.017</td>
<td>0.019</td>
</tr>
<tr>
<td>N20</td>
<td>CNC</td>
<td>3.313</td>
<td>3.209</td>
<td>2.410</td>
<td>1.359</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.021</td>
<td>0.016</td>
<td>0.017</td>
<td>0.019</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.021</td>
<td>0.016</td>
<td>0.017</td>
<td>0.019</td>
</tr>
<tr>
<td>BPI 2012</td>
<td>CNC</td>
<td>1.277</td>
<td>5.132</td>
<td>1.277</td>
<td>5.132</td>
</tr>
<tr>
<td></td>
<td>ACD</td>
<td>4.204</td>
<td>4.204</td>
<td>4.204</td>
<td>4.204</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.019</td>
<td>0.019</td>
<td>0.019</td>
<td>0.019</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.028</td>
<td>0.028</td>
<td>0.028</td>
<td>0.028</td>
</tr>
<tr>
<td>Hospital 1</td>
<td>CNC</td>
<td>3.256</td>
<td>3.256</td>
<td>3.256</td>
<td>3.256</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.028</td>
<td>0.028</td>
<td>0.028</td>
<td>0.028</td>
</tr>
<tr>
<td>Hospital 2</td>
<td>CNC</td>
<td>3.256</td>
<td>3.256</td>
<td>3.256</td>
<td>3.256</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>0.028</td>
<td>0.028</td>
<td>0.028</td>
<td>0.028</td>
</tr>
</tbody>
</table>

Table 2: Structural complexity measurements for the first and the second experiment.
(Mann-Whitney test: $U = 394.0 \ z = 2.186, \ p = 0.029 > 0.01$).

*Time performance* In both experiments, the technique took on average 27 secs to filter the log, with a minimum time of 0.05 secs and a maximum time of 279 secs. These performances are well within reasonable bounds.

5 Conclusion

We contributed a technique for the automatic removal of noise from process execution logs. The core idea of this technique is to use infrequent direct follows dependencies between event labels as a proxy for noise. These dependencies are detected and removed from an automaton built from the event log, and then transferred to the original log in the form of individual events to be removed, via alignment [2].

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, appropriateness and complexity without a significant negative effect on generalization. Further, the technique generally performs within seconds, with the worst reported case being below 5 minutes. Thus, this technique provides clear advantages over manual data cleaning – a challenging and time consuming task [17].

By relying on alignment, the technique guarantees that the number of events being removed from the original log is minimal, given a set of infrequent event dependencies. However, it does not guarantee that the set of such dependencies is maximal, i.e. there might be infrequent dependencies that are not detected in the first place. The problem is due to the interplay between the edges of the automaton whereby removing an edge may prevent some other edges from being removed in order to preserve the graph reachability. In future, we plan to explore the applicability of graph rewriting techniques (e.g. collapsing strongly-connected components) in combination with transitive reduction, in order to identify an optimal combination of infrequent dependencies to be removed. Another avenue for future work is to consider other types of event dependencies, e.g. transitive ones. Finally, we plan to work on automatically determining the best threshold to use for outliers detection in a given log. For example, this could be a function of the mean frequency of the dependencies in the log, rather than the general-purpose threshold of 5% used in statistics.

Acknowledgments. NICTA is funded by the Australian Government via the Department of Communications. This research is funded by the Australian Research Council Discovery Project DP150103356.

References