Timestamp Repair for Business Process Event Logs

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

Abstract—This paper contributes an approach for automatically correcting timestamp errors in business process execution logs. These errors are quite common in practice due to the logging granularity or the performance load of the logging system. Analyzing logs that have not been properly screened for such problems is likely to lead to wrong or misleading process insights. The proposed approach revolves around two techniques: one to reorder events with erroneous timestamps, the other to assign an estimated timestamp to each such event. The approach has been implemented in a software tool and extensively evaluated in different settings, using both synthetic and real-life logs. The experiments show that the approach significantly reduces the amount of incorrect timestamps, while the reordering of events scales well to large and complex datasets. The evaluation is complemented by a case study in the meat & livestock domain showing the usefulness of the approach in practice.

Index Terms—Process mining, data pre-processing, data cleaning, event log, timestamp repair.

1 Introduction

It is very frequent for real-world data to contain inaccurate or corrupt data records [15]. For example, a common problem in *event logs*, i.e. data records related to the execution of business processes, is that of incorrect timestamps. Each event in an event log captures the start or completion of a process activity and as such has a timestamp (e.g. activity "Reject purchase order" was completed at 12:30:00pm of 23/10/2017). A type of timestamp error is when a set of events related to the same process instance have the same timestamp. This error is either due to the granularity of the logging system, e.g. the system does not record the time but only the date, or to delays during the logging process, e.g. due to the logging system being overloaded.

Analyzing data that has not been properly screened for such problems is likely to lead to wrong or misleading insights. For example, in the context of automated process discovery [3], a class of process mining techniques [30] that aims to extract a process model from an event log, a common problem is to mix up order dependencies between process activities, due to timestamp errors: given that activities A and B have the same timestamp they are assumed to be concurrent, i.e. they can be executed in any order in the resulting process model, while in reality A is causal to B, i.e. A must always precede B in any instance of the process model. Clearly, any insights derived from a process model whose activities have wrong order dependencies, will be misleading.

While different approaches have been proposed to filter out events based on different characteristics, such as low frequency or the value of specific attributes deemed to be spurious, both in the context of event logs (e.g. [32], [7], [10]) and event streams (e.g. [34]), to the best of our knowledge, timestamp errors in event logs have been neglected.

In this paper we contribute an automated approach to repair likely timestamp errors in event logs by establishing a proper order among incorrectly-ordered events. Specifically, we address the situation where multiple events belonging to the same process instance (called *trace*) have the same timestamp. Our approach to tackle the "same-timestamp" error relies on two techniques. The first technique estimates, for each trace, the most-likely order between the events affected by the same-timestamp problem, using information from correctly-ordered events in the log, and repairs the log accordingly. The second technique assigns a timestamp to each event that has been reordered, based on an estimation of the duration of each process activity in the log.

We implemented our approach as a standalone tool as well as a plugin for two community-based open-source platforms for process mining. Using the standalone tool, we conducted a series of experiments with synthetic and real-life logs, aimed at measuring the accuracy and time performance of our approach in correcting same-timestamp errors, under different settings. In addition, we carried out a case study with an Australian organization in the meat & livestock domain to assess the usefulness of our approach in practice.

Against this backdrop, the rest of this paper is organized as follows. Section 2 discusses data cleaning approaches in the field of process mining and more generally in data mining. Section 3 presents the overall approach while Section 4 discusses the specific problem of finding the most likely sequence of events, and its formulation as an Integer Linear Programming problem. Next, Section 5 presents the experimental evaluation while Section 6 reports the results of the case study. Finally, Section 7 concludes the paper and discusses future work.

- *R. Conforti and M. La Rosa are with the University of Melbourne, Australia*
  Email: {raffaele.conforti, marcello.larosa}@unimelb.edu.au
- *Arthur H.M. ter Hofstede is with the Queensland University of Technology, Australia*
  Email: a.terhofstede@qut.edu.au

2 Related Work

Data pre-processing or data preparation [25] is that phase of a process or data mining project where raw data is transformed into an understandable format, so that it can be used as input for knowledge discovery techniques. A key step of data pre-processing is *data cleaning*, which involves the detection and correction of recording errors. Data cleaning has been dealt with from different perspectives in the field of process mining and more generally, in that of data mining. In the following, we first review approaches in these two fields and at the end we discuss their relevance to our work.

2.1 Data cleaning in process mining

Process mining aims at automatically extracting actionable process knowledge from event logs recording business process executions. This knowledge may take the form of a process model, performance measurements, a conformance report with respect to a normative model, and others. Each event in an event log contains information about what process activity has been performed (e.g. "Check invoice"), when it was performed (i.e. the timestamp of the activity's start or completion), and in which instance of the process (i.e. the case identifier, e.g. the order number or the claims number). Events within an event logs can be grouped in a trace, which is a sequence of events, generally ordered by their timestamp, all belonging to the same process instance.

Event log cleaning, often referred to as event log filtering, is a de-facto practice performed before starting any type of process mining analysis. The process of filtering an event log is generally an unstructured process as different event logs are affected by different issues. In an attempt to structure what is generally performed as an ad-hoc task, Suriadi et al. [29] identify a collection of eleven imperfection patterns that commonly affect the quality of an event log.

The ProM Framework [31] offers several plugins for log filtering. The *Filter Log using Simple Heuristics (SLF)* plugin is used to remove traces that do not start and/or end with specific events, as well as to remove events that refer to a specific process task or events that refer to an infrequent process task. Similarly, the *Filter Out Low-Frequency Traces* plugin removes traces that do not appear frequently enough, where the minimum frequency is determined by a user-defined threshold.

The *Filter Log using Prefix-Closed Language (PCL)* plugin removes events from traces to conform the log to a Prefix-Closed Language. A language is *prefix-closed* if the language is equal to the prefix closure of the language itself. For example, given a language $L = \{abc\}$, the prefix closure of L is defined as $Pref(L) = \{\epsilon, a, ab, abc\}$. Specifically, this plugin keeps a trace only if it is the prefix of another trace in the log.

Other log filtering plugins available in ProM offer the possibility of removing events that do not have a certain attribute, or where the attribute value does not match a given value set by the user, i.e. the *Filter Log by Attribute* plugin, or the possibility to extract a sublog including only the events related to a given timeframe (e.g. the first six months of recordings), i.e. the *Filter on Timeframe* plugin.

In the literature, log filtering is addressed from a noise-oriented viewpoint. Three approaches are available in this respect [32], [7], [10], [34]. The approach proposed by Wang et al. [32] uses a reference process model to repair a log whose events are affected by inconsistent labels, i.e. labels that do not match the expected behavior of the reference model. The approach proposed by Conforti et al. [7] removes events that cannot be reproduced by an automaton that is built from the process behavior recorded in the log, stripped from infrequent arcs. The latter approach is available as a plugin of the Apromore process analytics platform [18]. The approach by Fani Sani et al. [10] uses conditional probabilities between sequences of activities to remove events that are unlikely to occur in a given sequence. Finally, Zelst et al. [34] proposed an approach based on probabilistic automata for filtering out infrequent events from event streams of business processes.

2.2 Data cleaning in data mining

Data cleaning is commonly performed in the data mining field with the intent of isolating and removing outliers. Several outlier detection algorithms have been proposed in the literature. To identity outliers, these algorithms build a data model (e.g. a statistical, linear, or probabilistic model) which describes the normal behavior, and consider as outliers all data points which deviate from such a model [2].

In the context of temporal data, these algorithms have been surveyed by Gupta et al. [12] for events with continuous values, i.e. *time series*, and by Chandola et al. [6] for events with discrete values, i.e. *discrete sequences*.

These approaches can be classified into three major groups [12]. In the first group we have approaches that aim to detect if an entire sequence of events is an outlier. This is achieved either using the entire dataset to build a model (e.g. [5], [11], [35], [27]), or by subdividing the dataset into overlapping windows and building a model for each window (e.g. [19], [14], [20]). The second group contains approaches that detect if single data points (e.g. [8], [4], [24]) or sequences thereof (e.g. [33], [16]) are outliers on the basis of a data model of the normal behavior, e.g. a statistical model. Finally, approaches in the third group detect anomalous patterns within sequences (e.g. [13], [17]). These approaches assign an anomaly score to a pattern based on the difference between the frequency of the pattern in the training dataset and the frequency of the pattern in the sequence under analysis.

Independently of the field of origin (process mining or data mining), all the above are complementary works to ours, since they address different data cleaning problems than ours. These problems are related to the removal of undesired data, such as infrequent process behavior (noise, outliers) or particular event labels. In contrast, our goal is to establish a proper order among incorrectly-ordered events rather than removing them.

3 Approach

In this section, we present our approach for repairing event logs that contain traces where multiple events have the same timestamp and may, as such, be incorrectly ordered. First, we introduce preliminary concepts such as event log,

directly-follows dependencies, and log automaton. Next, we formalize the concept of incorrectly-ordered events at the level of each log trace. Finally, using this concept, we present a technique to identify the correct order between events for each such trace, and repair the trace accordingly, and a second technique to assign a timestamp to each event that has been reordered within the trace.

3.1 Preliminaries

Organizations often keep a record of the execution of their business processes, such as order-to-cash or procure-to-pay processes, in the form of event logs, which can later be used for process mining analyses. An *event log* is a set of *traces* where each trace (a.k.a. *case*) captures the execution of a particular process instance. Traces are recorded as sequences of events and are identified by a unique case identifier (case id). Each *event* refers to the execution of a specific process task within a trace at a specific time. For example, the event labeled "Invoice released" with case id "134" and timestamp "12-02-2017T12:30:00" means that an invoice has been released for case 134 (probably the order number) at 12:30 of 12 February 2017.

Definition 1 (Event Log)**.** *Let Γ be a finite set of tasks. A log $\mathcal{L}$ over Γ is defined as $\mathcal{L} \triangleq (\mathcal{E}, \mathcal{C}, C, T, \mathcal{T}, <)$ where $\mathcal{E}$ is the set of events, $\mathcal{C}$ is the set of case identifiers, $C : \mathcal{E} \to \mathcal{C}$ is a surjective function linking events to cases, $T : \mathcal{E} \to \Gamma$ is a surjective function linking events to tasks, $\mathcal{T} : \mathcal{E} \to \mathbb{N}$ is a surjective function linking events to timestamps, 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 $e_1 < e_2 \wedge C(e_1) = C(e_2) \wedge \nexists e_3 \in \mathcal{E}[C(e_3) = C(e_1) \wedge e_1 < e_3 \wedge e_3 < e_2]$.

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

Definition 2 (Directly-Follows Dependency)**.** *Given a log $\mathcal{L}$ and two tasks $x, y \in \Gamma$, x directly follows y, i.e. $x \rightsquigarrow y$, iff $\exists e_1, e_2 \in \mathcal{E} \mid T(e_1) = x \wedge T(e_2) = y \wedge e_1 \sqsubset e_2$.*

The directly-follows dependencies of a log can be represented using a directed graph where each node (here referred to as a state) represents a task which occurs in the log under consideration and each arc connecting two states indicates the existence of a directly-follows dependency between the corresponding tasks. From here on, this graph is referred to as a *log automaton*.

For convenience, the formal notation used so far as well as that introduced in the rest of this section, is summarized in Table 1.

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

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

Depending on the quality of the log, a log automaton may contain several rarely used directly-follows dependencies. To improve the quality of the log automaton these directly-follows dependencies can be removed using approaches such as the one proposed by Conforti et al. [7], which identifies and removes infrequent directly-follows dependencies. An automaton which has been filtered from infrequent directly-follows dependencies is referred to as an *anomaly-free* automaton, and it is defined as $\mathcal{A}^f \triangleq (\Gamma, \rightsquigarrow^r)$, where $\rightsquigarrow^r \subseteq \rightsquigarrow$ is the set of retained directly-follows dependencies.

Symbol	Meaning
Γ	Finite set of Tasks
$\mathcal{E}$	Finite set of Events
$\mathcal{C}$	Finite set of Trace Identifiers
C	Surjective function linking $\mathcal{E}$ to $\mathcal{C}$
T	Surjective function linking $\mathcal{E}$ to Γ
$\mathcal{T}$	Surjective function linking $\mathcal{E}$ to $\mathbb{N}$ (timestamps)
$<$	Strict total ordering over Events
$\sqsubset$	Strictly Before Relation
$\rightsquigarrow$	Directly-Follows Dependency
$\mathcal{A}$	Log Automaton
$\rightsquigarrow^r$	Set of Retained Directly-Follows Dependencies
$\mathcal{A}^f$	Anomaly-Free Automaton
$\dot{\sim}$	Timestamp Equivalence Relation
E	Set of Incorrectly Ordered Events
$\rightsquigarrow_i$	Set of Incorrect Directly-Follows Dependencies
$\rightsquigarrow_c$	Set of Correct Directly-Follows Dependencies
$\mathcal{A}^c$	Correct Log Automaton
$\mathcal{A}^{cf}$	Correct Anomaly-Free Automaton
$\#$	Function counting the occurrences of $\rightsquigarrow_c$
$\%$	Function the confidence level of $\rightsquigarrow_c$
$\mathcal{S}$	Set of possible sequences

TABLE 1: Formal notation.

3.2 Incorrect event order repair

Armed with the above notions, in this and in the next subsection we propose a technique for redefining a total order among events having the same timestamp. For simplicity, we assume that the events belong to a single trace, given that we are interested in the total order among events belonging to the same trace.

Definition 4 (Timestamp Equivalent Events)**.** *Given a log $\mathcal{L}$ and two events $e_1, e_2 \in \mathcal{E}$, e_1 is timestamp equivalent to e_2, i.e. $e_1 \dot{\sim} e_2$, if and only if they belong to the same trace and they have the same timestamp, i.e. $e_1 \dot{\sim} e_2$ iff $C(e_1) = C(e_2) \wedge \mathcal{T}(e_1) = \mathcal{T}(e_2)$.*

We use this equivalence relation to partition the log in sets of events that are timestamp equivalent. We refer to these sets of timestamp equivalent events as the set of *incorrectly ordered events*.

Definition 5 (Incorrectly Ordered Events)**.** *Given a log $\mathcal{L}$ the set of incorrectly ordered events E is the quotient set of $\mathcal{E}$ by $\dot{\sim}$. Formally, E is defined as $E \triangleq \mathcal{E}/\dot{\sim}$.*

The presence of such incorrectly ordered events, may be the cause for *incorrect directly-follows dependencies*. A directly-follows dependency is incorrect if it is caused by an incorrectly ordered event, i.e. the directly-follows dependency is not observed if all incorrectly ordered events are removed.

Definition 6 (Incorrect Directly-Follows Dependency)**.** *Given two tasks $x, y \in \Gamma$, x incorrectly directly follows y, i.e. $x \rightsquigarrow_i y$, iff $x \rightsquigarrow y \wedge \nexists e_1, e_2 \in \mathcal{E} \mid T(e_1) = x \wedge T(e_2) = y \wedge C(x) = C(y) \wedge e_1 \sqsubset e_2 \wedge \mathcal{T}(e_1) \neq \mathcal{T}(e_2)$.*

Additionally, we define the set of *correct directly-follows dependencies* as $\rightsquigarrow_c \triangleq \rightsquigarrow \setminus \rightsquigarrow_i$, and the automaton constructed over this set as *correct automaton*, i.e. $\mathcal{A}^c = (\Gamma, \rightsquigarrow_c)$.

When attempting to fix incorrectly ordered events, we wish to propose a solution that is free from infrequent behaviour as well as free from incorrect event ordering. This can be achieved using the correct log automaton and the anomaly-free automaton. To guarantee that both the requirements imposed by the two automata are satisfied, we define the *correct anomaly-free automaton* as the automaton defined over the intersection of the two automata.

Definition 7 (Correct Anomaly-Free Automaton)**.** *Given an anomaly-free automaton $\mathcal{A}^f \triangleq (\Gamma, \rightsquigarrow^r)$ defined over event log $\mathcal{L}$, and a correct log automaton $\mathcal{A}^c \triangleq (\Gamma, \rightsquigarrow_c)$ defined over the same event log, a* correct anomaly-free automaton *is defined as $\mathcal{A}^{cf} \triangleq (\Gamma, \rightsquigarrow^r_c)$, where $\rightsquigarrow^r_c \triangleq \rightsquigarrow_c \cap \rightsquigarrow^r$.*

Additionally, we introduce the function $\# : \Gamma \times \Gamma \to \mathbb{N}$ which retrieves the number of times a directly-follows dependency of a correct anomaly-free automaton occurs in the underlying log. Formally, given two tasks x and y, $\#(x, y)$ is defined as $\#(x, y) \triangleq |\{(e_1, e_2) \in \mathcal{E} \times \mathcal{E} \mid x \rightsquigarrow^r_c y \wedge e_1 \sqsubset e_2 \wedge T(e_1) = x \wedge T(e_2) = y\}|$.

Using the frequency of a directly-follows dependency, we introduce the function $\% : \Gamma \times \Gamma \to [0, 1]$, which provides an estimate of the confidence level of a directly-follows dependency in a correct anomaly-free automaton. This function is defined as:

$$\%(x, y) \triangleq \frac{\#(x, y)}{\sum_{z \in \Gamma} \#(x, z)} \tag{1}$$

A correct anomaly-free automaton is used to identify the optimal order among events having the same timestamp. This is achieved selecting the most likely sequence of events among all possible sequences. Given a set of events $\omega \in \mathcal{P}(\mathcal{E})$, the set of all possible sequences that can be generated is defined as:

$$\mathcal{S}(\omega) \triangleq \{\overline{\omega} \in \mathcal{E}^* \mid |\overline{\omega}| = |\omega| \wedge \forall e \in \omega \; \exists 1 \leq i \leq |\omega| \, [e = \overline{\omega}_i]\} \tag{2}$$

Using the confidence level of a directly-follows dependency, we can measure the confidence level of a sequence through the function $\% : \mathcal{E}^* \to [0, 1]$. Although several implementations of this function can be defined, at this stage we are purposely leaving it unspecified. In the following sections, we will discuss in more details some possible implementations and related implications.

The confidence level of a sequence is ultimately used to identify the most likely sequence over a set of events. Formally, given a set of events $\omega \in \mathcal{P}(\mathcal{E})$, we define the sequence with the highest confidence level as $\overline{\omega}^b \in \mathcal{S}(\omega) \mid \nexists \, \overline{\omega} \in \mathcal{S}(\omega)[\%(\overline{\omega}) > \%(\overline{\omega}^b)]$.

Algorithm 1 shows how to repair a log $\mathcal{L}$ affected by incorrectly ordered events using the elements defined so far. For each trace in the log (see line 1) it retrieves the set of incorrectly ordered events belonging to such a trace (see line 2). It then performs two major operations over each set of events which may have been ordered incorrectly. Firstly, it removes the strict total ordering defined among each couple of events in the set (see line 4). Once the strict total ordering involving these events has been removed, it introduces a new total ordering based on the sequence of events with the highest confidence level (see line 6). Finally, it returns the modified log.

Algorithm 1: RepairLog

```
   input: Event log L
1  for c ∈ C do
2      for ω ∈ E do
3          for (e1, e2) ∈ < do
4              if e1 ∈ ω ∧ e2 ∈ ω then <⇐< \{(e1, e2)}
               ;
5          for i ← 1 to |ω^b| − 1 do
6              <⇐< ∪{(ω^b_i, ω^b_{i+1})}
7  return L
```

3.3 Events sequence confidence level

In Section 3.2, we discussed the possibility of having different implementations for the function $\% : \mathcal{E}^* \to [0, 1]$. Following a purely probabilistic approach the first implementation to come to mind would be something like:

$$\%(\overline{\omega}) \triangleq \prod_{i=1}^{|\overline{\omega}|-1} \%(T(\overline{\omega}_i), T(\overline{\omega}_{i+1})) \tag{3}$$

where the confidence level of the entire sequence is based on the likelihood of pairs of events following each other. While in theory this would work, in practice this implementation has a major fault. Let us assume that we have a set of three events e_1, e_2, and $e_3 \in \mathcal{E}$ all with the same timestamp, with $T(e_1) = a, T(e_2) = b$, and $T(e_3) = c$. Moreover, let us assume that only the following directly-follows dependencies hold: $a \rightsquigarrow b$ (with a confidence level of 0.5) and $a \rightsquigarrow c$ (with a confidence level of 0.1). In this case, all possible sequences that can be generated out of those three events will have a confidence level of 0 (because $b \rightsquigarrow c$ is missing).

Considering the confidence level of each directly-follows dependency, a sequence where event e_1 is directly followed by event e_2 may be preferred over other possible sequences. In this case we may consider for example sequence $\langle e_1, e_2, e_3 \rangle$ or sequence $\langle e_3, e_1, e_2 \rangle$ as the most likely to exist since they contain the sequence $\langle e_1, e_2 \rangle$. Following this reasoning the function $\%' : \mathcal{E}^* \to \mathbb{R}$ may be preferred for the identification of the most likely sequence of events.

$$\%'(\overline{\omega}) \triangleq \sum_{i=1}^{|\overline{\omega}|-1} \%(T(\overline{\omega}_i), T(\overline{\omega}_{i+1})) \tag{4}$$

Figure 1 offers a visual aid of how the approach works in the case of a log with its last three traces affected by incorrectly-ordered events. Specifically, we have the events $[B, C, D]$, $[C, D, B]$, and $[C, D]$ having the same timestamp in their respective traces.

The first step, is the discovery of the log automaton. To improve readability we annotated each node and arc of the automaton with the number of times the node is visited or the arc is traversed. From the figure we can observe that the automaton has several undesired arcs which are either

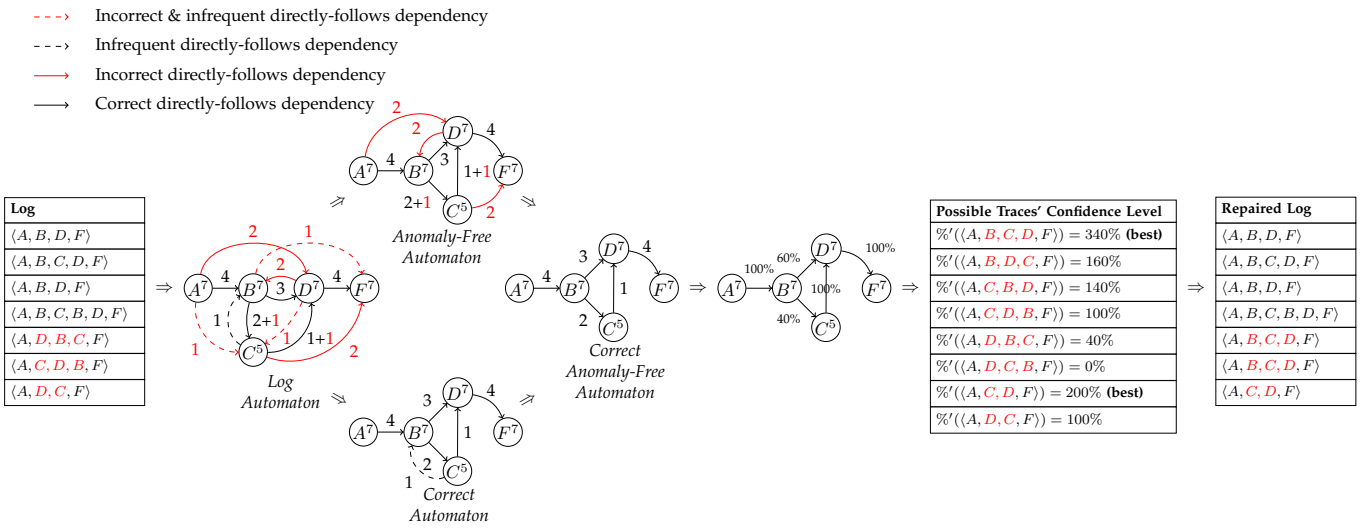


Fig. 1: From noise log to repaired log, using the Correct Anomaly-Free Automaton.

infrequent, incorrect, or both. This automaton is filtered to generate the anomaly-free and the correct automaton. It is important to note how those two automata filter out different arcs (e.g. the correct automaton retains arc (c, b) and removes arc (c, f), while the anomaly-free automaton does the opposite).

In the case of the anomaly-free automaton we filtered out all the arcs that had a frequency below 0.2.[1] As a result, we removed the arcs $(a, c), (b, f), (c, b)$, and (d, c). On the other hand, in the correct automaton we removed all arcs that were generated exclusively by incorrectly ordered events, i.e. the arcs $(a, c), (a, d), (b, f), (c, f), (d, b)$, and (d, c).

In the following step, we merge the two automata into the correct anomaly-free automaton, retaining only the arcs that are common to both automata. This automaton is then used to compute the confidence level of possible traces. The measurement of the confidence level of a trace relies on the confidence level of each arc of the automaton. Given an arc a, the confidence level of a is computed as the number of times a is traversed divided by the sum of the number of times each arc originated from the same source of a is traversed, e.g. (b, d) has a confidence level of 60% as there are $2 + 3 = 5$ ways to leave node b and 3 are through the arc (b, d).

The final step is the computation of the confidence level of each possible trace and the consequent selection of the best one. The confidence level of a trace is computed by summing the confidence level of each couple of subsequent events. For example in case of a trace $\langle A, B, C, D, F\rangle$, we have to consider the confidence level of the following couples $[A, B]$, $[B, C]$, $[C, D]$, and $[D, F]$. This leads us to a confidence level of $\#(A, B) + \#(B, C) + \#(C, D) + \#(D, F) = 100\% + 40\% + 100\% + 100\% = 340\%$, which marks this sequence as the most likely out of all possible permutations of trace $\langle A, B, C, D, F\rangle$.

1. The frequency of an arc is measured as twice the number of times the arc is traversed divided by the sum of the number of times the source node and target node of the arc are visited.

3.4 Event timestamp estimation

The second technique in our approach assigns a timestamp to each ordered event. In order to estimate an event's timestamp, we rely on the distribution of the durations of each activity, i.e. their duration *probability density functions* (PDFs).

The first step consists in estimating the duration of each event. Given an event $e_1 \in \mathcal{E}$, the duration of e_1 is estimated by subtracting the timestamp of the event preceding e_1 from the timestamp of e_1. As part of this paper we estimate the duration of an event using the surjective function $\mathcal{D}$. Formally, we define $\mathcal{D} : \mathcal{E} \to \mathbb{N}$ as:

$$\mathcal{D}(e_1) \triangleq \begin{cases} \mathcal{T}(e_1) - \mathcal{T}(e_2) & \text{if } e_2 \in \mathcal{E} \text{ such that } e_2 \rightsquigarrow_c e_1, \\ \epsilon & \text{otherwise} \end{cases} \tag{5}$$

Where ϵ is an arbitrary small value to avoid that when assigning a timestamp we will cause two events to be timestamp equivalent.

For a given activity $t \in \Gamma$ we can define the multi-set of its events' durations as $\mathcal{D}_t \triangleq \{(d, n) \in \mathbb{N} \times \mathbb{N} \mid \exists e \in \mathcal{E}_{t,d} \wedge n = |\mathcal{E}_{t,d}|\}$ where $\mathcal{E}_{t,d}$ is the subset of $\mathcal{E}$ containing only events of activity t that have a given duration d, i.e. $\mathcal{E}_{t,d} \triangleq \{e \in \mathcal{E} \mid T(e) = t \wedge d = \mathcal{D}(e)\}$.

We use this multiset to estimate the PDF of the duration of an activity. For this purpose we define the function estimating the duration of an activity a as $f_a^{est} : \mathbb{N} \times \mathbb{N} \to (\mathbb{N} \to \mathbb{R})$, where the range is the PDF (f_a) we want to estimate. This can be achieved using approaches as the one proposed by Silverman [26] for the estimation of multimodal distributions using kernel density.

Once the PDF of each activity is known, we can assign a new timestamp to each ordered event following the durations dictated by the PDF. The new timestamp is obtained as:

$$\mathcal{T}^+(e) \triangleq \begin{cases} \mathcal{T}(e_1) + \overline{f_{T(e)}} & \text{if } e_1 \in \mathcal{E} \text{ such that } e_1 \rightsquigarrow_c e, \\ \epsilon + \overline{f_{T(e)}} & \text{otherwise} \end{cases} \tag{6}$$

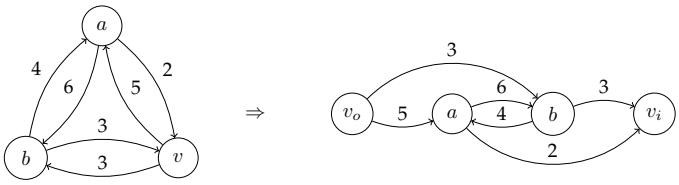


Fig. 2: Sample Reduction from Max TSP to Most Likely Sequence of Events Problem.

where $\overline{f_{T(e)}}$ extracts a likely duration from the PDF $f_{T(e)}$. Finally, the new timestamp for event e substitutes the old one in the log, i.e. $\mathcal{T} \Leftarrow \mathcal{T} \oplus \{(e, \mathcal{T}^+(e))\}$.

4 Most likely sequence of events Problem

In this section first we discuss the inherent complexity of determining the most likely sequence of events and then we provide an ILP formulation of the problem.

4.1 Complexity

The identification of the most likely sequence of events is an NP-hard problem. In this section we provide a sketch of its complexity by providing a polynomial time transformation from the Maximum Travelling Salesman Problem (Max TSP) (a well known NP-complete problem [22]) to the most likely sequence of events problem.

The Max TSP is the problem of identifying a Hamiltonian cycle (a tour that passes through all the vertices) with maximum cost in a complete asymmetric graph with non-negative weights. It is straightforward to show that a Max TSP problem can be reduced to an instance of the most likely sequence of events problem. This can be achieved through the application of the following six polynomial steps:

- select a random vertex v of the Max TSP problem;
- introduce two new vertices v_i and v_o;
- for each incoming arc of v, i.e. $a = (s, v)$, create a new arc $a_i = (s, v_i)$;
- for each outgoing arc of v, i.e. $a = (v, t)$, create a new arc $a_o = (v_o, t)$;
- remove all arcs connected to v;
- remove vertex v;

Now the Max TSP problem corresponds to determining the most likely sequence of events leading from v_o to v_i.

Figure 2 shows how to reduce a Max TSP problem to a most likely sequence of events problem. In the example, the asymmetric graph contains three vertices, $V = \{a, b, v\}$. These three vertices are connected via six weighted arcs $A = \{(a, b, 6), (a, v, 2), (b, a, 4), (b, v, 3), (v, a, 5), (v, b, 3)\}$. As part of the reduction two new vertices v_i and v_o are introduced, as well as arcs connecting these vertices to vertices which were originally connected to v.

4.2 ILP formulation

In section 3.2, we defined $\overline{\omega}^b$ as the most likely sequence over the set $\mathcal{S}(\omega)$ of possible sequences. To determine the most likely sequence over the set $\mathcal{S}(\omega)$ we propose the application of Integer Linear Programming (ILP), where the confidence level of a sequence is measured using the function proposed in equation 4.

Given a log $\mathcal{L}$, the inputs for our problem are the set of events $\omega \in E$, the event $s \in \mathcal{E}$ (not affected by timestamp error) which should precede the sequence $\overline{\omega}^b$, and the event $f \in \mathcal{E}$ (not affected by timestamp error) which should follow the sequence $\overline{\omega}^b$. Finally, for convenience we define the set $\omega^f \triangleq \omega \cup \{f\}$, and $\omega^{fs} \triangleq \omega \cup \{s, f\}$.

In our approach we model the problem of determining the best sequence as a traveling salesman problem (TSP) [23]. Our formulation presents two main differences from the original formulation. Firstly, we try to maximize the distance instead of minimizing it (which corresponds to a change in equation 7). Secondly, we impose that f must close the cycle on s (which corresponds to equation 8). Equations 9, 10, and 11 are directly from [23].

Before presenting the formulation the following set of variables needs to be introduced:

- for each event $e \in \omega^{fs}$ there exists an auxiliary variable $u_e \in \mathbb{Z}$.
- for each couple of events $(e_1, e_2) \in \omega^{fs} \times \omega^{fs}$ if $e_1 \neq e_2$ there exists a variable $x_{e_1,e_2} \in \{0, 1\}$. If the solution of the ILP problem is such that $x_{e_1,e_2} = 1$, event e_2 directly follow event e_1 in the most likely sequence.

The ILP problem aims at maximizing the confidence level of the sequence as proposed in equation 4:

$$\max \sum_{e_1 \in \omega^{fs}} \sum_{e_2 \in \omega^{fs}} \%(T(e_1), T(e_2)) \cdot x_{e_1,e_2}. \tag{7}$$

This ILP problem is subject to the following constraints:

- We connect the final event to the start event to prevent them from being in the middle of the sequence:

$$x_{f,s} = 1. \tag{8}$$

- We force each event $e_1 \in \omega^f$ to precede one and only one event:

$$\sum_{e_2 \in \omega^{fs} \setminus \{e_1\}} x_{e_1,e_2} = 1. \tag{9}$$

- We force each event $e_1 \in \omega^f$ to follow one and only one event:

$$\sum_{e_2 \in \omega^{fs} \setminus \{e_1\}} x_{e_2,e_1} = 1. \tag{10}$$

- For each couple of events $(e_1, e_2) \in \omega^f \times \omega^f$ where $e_1 \neq e_2$:

$$u_{e_1} - u_{e_2} + \left|\omega^{fs}\right| \cdot x_{e_1,e_2} \leq \left|\omega^{fs}\right| - 1 \tag{11}$$

With equation 11, we prevent the presence of sub-tours in the TSP problem, i.e. we enforce the tour to visit all cities in one go. In the context of our problem this means that we enforce the presence of a single sequence covering all events.

5 Evaluation

We conducted two experiments to assess the accuracy of our approach using a combination of synthetic and real-life logs. To perform these experiments we implemented the technique in Java,[2] and released it as a standalone tool,[3] as well as a plugin for the Apromore process analytics platform,[4] and for the ProM Framework.[5]

5.1 Design

To measure the accuracy of our approach we designed two experiments. In the first experiment, we used a seed model to generate several logs affected by different levels of *incorrectly-ordered events*. Using our approach we repaired these logs and measured the accuracy of the repair.

In the second experiment, we used a real-life log affected by incorrectly-ordered events for which the correct order of the events is known. After repairing the log we measured the number of correctly repaired events. Figure 3 provides a schematic representation of the two experiments.

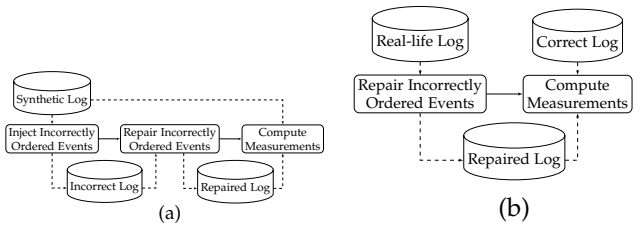


Fig. 3: Experiments setup for artificial (a) and real-life logs (b).

We measured accuracy using the *Levenshtein distance* [21] between the proposed repair and the correct log. The Levenshtein distance is a *string edit distance* which measures the difference between two strings by counting the number of single-character edits required to transform one string into another. The Levenshtein distance considers three types of single-character edits: i) the insertion of a character, ii) the removal of a character, and iii) the replacement/swap of a character. In our case, we adapted this *edit distance* to the concept of traces where instead of inserting, removing, or replacing a character we insert, remove, or replace an event. We decided to use this metric since it provides us with a fine-grained measurement which will allow us to assess the quality of a technique even in cases where the optimal solution is not identified.

Moreover, for the synthetic logs, we also measured the level of conformance between the repaired log and the model used to generate the logs, and the root-square mean error (RSME) between the assigned timestamp and the correct one. The level of conformance was measured using *alignment-based replay fitness* [1] which is an established measure of the quality of automatically discovered process models.

5.2 Datasets

The dataset for the first experiment is derived from the BPMN model shown in Figure 4. We used this model since it exhibits properties often found in models discovered from real-life logs, such as high degree of concurrency, unstructuredness, skips, and loops. Starting from a log obtained by simulating the model, we injected different levels of *incorrectly ordered events*. We injected incorrectly-ordered events at three levels of granularity (events, traces and unique traces) by changing the percentage of affected events, traces or unique traces in increments of 5%, ranging from 5% to 40%. This produced three sets of eight logs, for a total of 24 logs.

We performed the second experiment using the BPI Challenge 2014 log [9]. Out of all the publicly-available event logs in the 4TU Data Center,[6] this was the only log affected by incorrectly-ordered events for which the correct order could be retrieved. To retrieve the correct order of events, we rearranged the events in the log over the attribute *IncidentActivity_Number*. We used this attribute since it is present in each event and we know from the accompanying process description[7] that this attribute is incremented automatically at each event.

5.3 Results

Table 2 provides an overview of the logs used for our two experiments, reporting on the number of traces, the number of unique traces, the number of events, the number of unique labels, and the percentage of incorrectly ordered events.

Table 3 shows the results of the experiment performed over the first set of logs. Specifically, it reports number of traces affected by incorrect ordered events, absolute edit distance, maximum edit distance, average edit distance computed over the entire log and its standard deviation, average edit distance computed using only traces affected by incorrectly ordered events and its standard deviation, RMSE and fitness. The results show that our technique improved the quality of the log reducing its edit distance by 60% on average. Moreover, it outperformed both baseline techniques, performing 480 times better than the naïve repair and 4,700 times better than the random repair.

Figure 5 shows the plots of the cumulative edit distance, the average edit distance over the entire log, and the average edit distance over the affected traces only. Two things can be evinced from the graph. The first one is that the number of remaining incorrectly ordered events is proportional to the number inserted, independent of the technique used. The second and more interesting point has to do with the average edit distance over the affected traces only (see Figure 5c). Our technique is the only one to have an average sitting below the average of the affected log. This implies that the two baseline techniques are only repairing traces

2. The source code is available at https://github.com/raffaeleconforti/ResearchCode
3. http://apromore.org/platform/tools
4. http://www.apromore.org
5. http://www.promtools.org
6. https://data.4tu.nl/repository/collection:event_logs_real
7. https://www.win.tue.nl/bpi/doku.php?id=2012:challenge

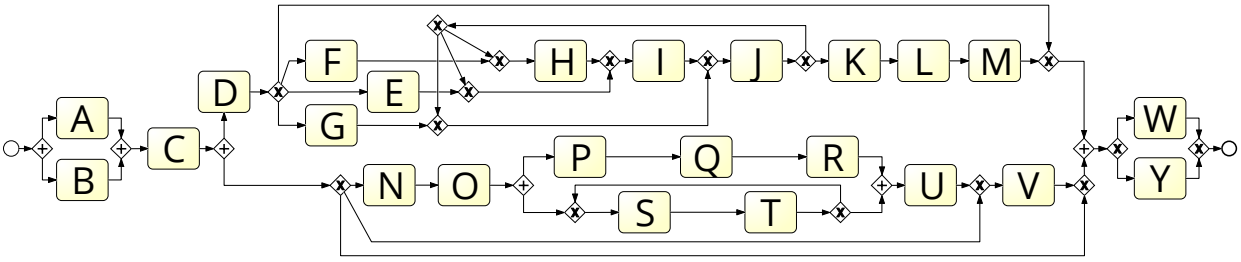


Fig. 4: Process Model of the Synthetic Log

Synthetic Log	#Traces	#Unique Traces	#Events	#Unique Labels	%Incorrect Events
Reference	3000	1277	45330	24	0%
$N5_e$	3000	1440	45330	24	5%
$N10_e$	3000	1591	45330	24	10%
$N15_e$	3000	1715	45330	24	15%
$N20_e$	3000	1855	45330	24	20%
$N25_e$	3000	1978	45330	24	25%
$N30_e$	3000	2110	45330	24	30%
$N35_e$	3000	2231	45330	24	35%
$N40_e$	3000	2361	45330	24	40%
$N5_t$	3000	1368	45330	24	5%
$N10_t$	3000	1453	45330	24	10%
$N15_t$	3000	1544	45330	24	15%
$N20_t$	3000	1632	45330	24	20%
$N25_t$	3000	1695	45330	24	25%
$N30_t$	3000	1772	45330	24	30%
$N35_t$	3000	1849	45330	24	35%
$N40_t$	3000	1924	45330	24	40%
$N5_{ut}$	3000	1391	45330	24	5%
$N10_{ut}$	3000	1434	45330	24	10%
$N15_{ut}$	3000	1506	45330	24	15%
$N20_{ut}$	3000	1559	45330	24	20%
$N25_{ut}$	3000	1616	45330	24	25%
$N30_{ut}$	3000	1659	45330	24	30%
$N35_{ut}$	3000	1738	45330	24	35%
$N40_{ut}$	3000	1863	45330	24	40%
Real-life Log	**#Traces**	**#Unique Traces**	**#Events**	**#Unique Labels**	**%Incorrect Events**
$BPI2014_{sorted}$	46616	18627	466737	39	0%
BPI2014	46616	22632	466737	39	39%

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

easy to repair, i.e. with an edit distance below average. Our technique on the other hand can repair complex traces, lowering the maximum edit distance by 12.5% on average.

Tables 4 and 5 show the results of the experiments over the second and third sets of synthetic logs. Over these two sets of logs, our technique again outperforms the other two baseline techniques. Interesting are the results obtained when incorrectly ordered events are inserted at unique trace level. In this case, the two baseline techniques are not able to repair the log. They reduce the cumulative edit distance on average by 0% and 5% (see Figures 6a and 6b) while our technique maintains an average of 60%.

Another important point emerging from the experiments is the accuracy of our approach when assigning a timestamp to an event. In the experiments with the three sets of logs our technique improved the RMSE on average by 54%, with a minimum of 48% (achieved over the most complex log, i.e. $N40_{ut}$) and a maximum of 59%. Additionally, it should be noted that the two baseline techniques only managed to achieve on average an improvement of 4% for the naïve technique and 0% for the random technique. Moreover, the level of conformance between a log and the reference model is the highest across all experiments when the repair is achieved using our technique.

Finally, Table 6 reports the results obtained using the real-life log. The results show that our technique noticeably outperforms the two baseline techniques improving the cumulative edit distance by 47%, while the two baselines techniques, i.e. naive repair and random repair, produced an improvement of 2% and 3% respectively. These results, comparable to the results obtained with the synthetic logs, confirm the efficacy of our technique and show its applicability in real-life scenarios.

Time Performance. Table 7 reports the time performance for the experiments with artificial and real-life logs, as the time required to reorder the events, the time required to estimate their timestamp, and the total time. These results show that the proposed technique for establishing an ordering between events is quite efficient (ranging from 3.5 to 7.8 sec on average); this time grows linearly with the number of events affected by timestamp issues. It must be noted that, while the amount of time required to repair the BPI2014 log may seem unreasonable (over 45 min), in this log there are over $180,000$ events (39% of the total number of events) affected by timestamp issues, which leads to an average of 15 msec per event.

Generally, the bulk of the time is taken by the timestamp estimation (ranging from 4.5 to 5.75 min on average for the

Log	Technique	# Affected Traces	Distance		Entire Log		Affected Traces		RMSE in Days	Fitness
			Cumulative	Max	Average Distance	Stand. Dev.	Average Distance	Stand. Dev.		
$N5_e$	Affected	273	1785	26	0.595	2.397	6.538	4.927	121.62	0.972
	Our	**143**	**729**	**23**	**0.243**	**1.438**	**5.098**	**4.318**	**42.65**	**0.999**
	Naïve	202	1481	26	0.494	2.306	7.332	5.372	114.27	0.981
	Random	249	1735	26	0.578	2.394	6.968	4.950	121.53	0.973
$N10_e$	Affected	531	3590	26	1.197	3.289	6.761	4.849	123.99	0.945
	Our	**286**	**1412**	**23**	**0.471**	**1.919**	**4.937**	**4.071**	**42.82**	**0.998**
	Naïve	394	2996	26	0.999	3.192	7.604	5.230	117.45	0.962
	Random	504	3533	26	1.178	3.290	7.010	4.852	123.94	0.946
$N15_e$	Affected	792	5349	26	1.783	3.868	6.754	4.808	122.78	0.920
	Our	**444**	**2185**	**23**	**0.728**	**2.342**	**4.921**	**4.052**	**43.16**	**0.997**
	Naïve	594	4546	26	1.515	3.801	7.653	5.100	117.15	0.941
	Random	747	5252	26	1.751	3.874	7.031	4.810	122.75	0.922
$N20_e$	Affected	1062	7102	26	2.367	4.326	6.687	4.897	123.58	0.894
	Our	**586**	**2971**	**24**	**0.990**	**2.744**	**5.070**	**4.228**	**44.28**	**0.996**
	Naïve	777	5960	26	1.987	4.290	7.671	5.240	117.77	0.924
	Random	997	6962	26	2.321	4.339	6.983	4.909	123.52	0.897
$N25_e$	Affected	1314	8885	26	2.962	4.668	6.762	4.904	123.98	0.867
	Our	**734**	**3742**	**24**	**1.247**	**3.027**	**5.098**	**4.221**	**44.83**	**0.995**
	Naïve	974	7504	26	2.501	4.680	7.704	5.233	118.29	0.905
	Random	1237	8723	26	2.908	4.689	7.052	4.909	123.94	0.871
$N30_e$	Affected	1600	10673	26	3.558	4.898	6.671	4.921	123.47	0.840
	Our	**876**	**4496**	**24**	**1.499**	**3.249**	**5.132**	**4.182**	**44.46**	**0.994**
	Naïve	1175	9004	26	3.001	4.981	7.663	5.256	117.87	0.886
	Random	1511	10490	26	3.497	4.929	6.942	4.931	123.45	0.843
$N35_e$	Affected	1881	12440	26	4.147	5.042	6.614	4.922	123.57	0.813
	Our	**1030**	**5226**	**24**	**1.742**	**3.416**	**5.074**	**4.134**	**43.87**	**0.993**
	Naïve	1374	10496	26	3.499	5.211	7.639	5.260	118.09	0.866
	Random	1756	12185	26	4.062	5.093	6.939	4.935	123.54	0.818
$N40_e$	Affected	2168	14177	30	4.726	5.073	6.539	4.874	123.34	0.785
	Our	**1174**	**5884**	**23**	**1.961**	**3.542**	**5.012**	**4.096**	**43.86**	**0.992**
	Naïve	1598	11969	30	3.990	5.339	7.490	5.224	117.94	0.847
	Random	2046	13926	30	4.642	5.132	6.806	4.888	123.29	0.791

TABLE 3: Results of the experiment with synthetic logs with incorrectly-ordered events inserted at event level.

Log	Technique	# Affected Traces	Distance		Entire Log		Affected Traces		RMSE in Days	Fitness
			Cumulative	Max	Average Distance	Stand. Dev.	Average Distance	Stand. Dev.		
$N5_t$	Affected	150	995	26	0.332	1.826	6.633	4.989	120.68	0.984
	Our	**75**	**398**	**22**	**0.133**	**1.092**	**5.307**	**4.496**	**51.92**	**0.999**
	Naïve	106	811	26	0.270	1.748	7.651	5.481	112.58	0.989
	Random	139	971	26	0.324	1.822	6.986	5.014	120.62	0.985
$N10_t$	Affected	300	1985	26	0.662	2.511	6.617	4.862	121.55	0.970
	Our	**158**	**800**	**22**	**0.267**	**1.485**	**5.063**	**4.193**	**50.86**	**0.999**
	Naïve	223	1647	26	0.549	2.415	7.386	5.286	114.29	0.979
	Random	284	1951	26	0.650	2.509	6.870	4.875	121.50	0.970
$N15_t$	Affected	450	2947	26	0.982	2.974	6.549	4.746	122.08	0.955
	Our	**235**	**1167**	**22**	**0.389**	**1.764**	**4.966**	**4.125**	**50.31**	**0.998**
	Naïve	331	2430	26	0.810	2.867	7.341	5.152	114.87	0.969
	Random	427	2901	26	0.967	2.974	6.794	4.750	122.05	0.956
$N20_t$	Affected	600	4043	26	1.348	3.444	6.738	4.793	123.63	0.938
	Our	**318**	**1584**	**23**	**0.528**	**2.022**	**4.981**	**4.047**	**51.00**	**0.998**
	Naïve	445	3382	26	1.127	3.351	7.600	5.150	117.09	0.956
	Random	560	3960	26	1.320	3.446	7.071	4.790	123.60	0.939
$N25_t$	Affected	750	5082	26	1.694	3.799	6.776	4.826	123.44	0.924
	Our	**417**	**2066**	**22**	**0.689**	**2.295**	**4.954**	**4.095**	**50.88**	**0.997**
	Naïve	564	4318	26	1.439	3.726	7.656	5.125	117.79	0.944
	Random	705	4990	26	1.663	3.804	7.078	4.821	123.41	0.926
$N30_t$	Affected	900	6065	26	2.022	4.087	6.739	4.887	123.65	0.909
	Our	**489**	**2475**	**24**	**0.825**	**2.526**	**5.061**	**4.208**	**51.65**	**0.996**
	Naïve	664	5105	26	1.702	4.027	7.688	5.221	117.74	0.934
	Random	842	5949	26	1.983	4.096	7.065	4.886	123.62	0.911
$N35_t$	Affected	1050	7038	26	2.346	4.316	6.703	4.901	123.81	0.895
	Our	**578**	**2928**	**24**	**0.976**	**2.727**	**5.066**	**4.229**	**52.07**	**0.996**
	Naïve	770	5906	26	1.969	4.277	7.670	5.248	117.97	0.925
	Random	985	6903	26	2.301	4.329	7.008	4.908	123.79	0.897
$N40_t$	Affected	1200	8031	26	2.677	4.504	6.692	4.883	123.90	0.880
	Our	**657**	**3317**	**24**	**1.106**	**2.868**	**5.049**	**4.202**	**51.08**	**0.995**
	Naïve	881	6741	26	2.247	4.492	7.652	5.230	118.06	0.914
	Random	1132	7891	26	2.630	4.521	6.971	4.889	123.87	0.882

TABLE 4: Results of the experiment with synthetic logs with incorrectly-ordered events inserted at trace level.

Log	Technique	# Affected Traces	Distance		Entire Log		Affected Traces		RMSE in Days	Fitness
			Cumulative	Max	Average Distance	Stand. Dev.	Average Distance	Stand. Dev.		
$N5_{ut}$	Affected	235	1253	22	0.418	1.878	5.332	4.340	109.84	0.978
	Our	**87**	**444**	**16**	**0.148**	**1.077**	**5.103**	**3.833**	**55.71**	**0.999**
	Naïve	235	1160	22	0.387	1.821	4.936	4.457	104.74	0.984
	Random	235	1252	22	0.417	1.878	5.328	4.343	109.82	0.978
$N10_{ut}$	Affected	363	1978	25	0.659	2.377	5.449	4.539	114.36	0.967
	Our	**151**	**761**	**19**	**0.254**	**1.402**	**5.040**	**3.865**	**58.06**	**0.999**
	Naïve	363	1849	25	0.616	2.319	5.094	4.652	110.99	0.974
	Random	363	1975	25	0.658	2.376	5.441	4.544	114.34	0.968
$N15_{ut}$	Affected	549	3092	25	1.031	2.956	5.632	4.672	114.92	0.949
	Our	**219**	**1181**	**19**	**0.394**	**1.777**	**5.393**	**4.039**	**57.98**	**0.999**
	Naïve	549	2946	25	0.982	2.911	5.366	4.773	112.68	0.958
	Random	549	3089	25	1.030	2.955	5.627	4.676	114.90	0.950
$N20_{ut}$	Affected	674	3895	25	1.298	3.288	5.779	4.715	116.26	0.950
	Our	**305**	**1587**	**21**	**0.529**	**2.028**	**5.203**	**4.016**	**59.31**	**0.998**
	Naïve	674	3736	25	1.245	3.250	5.543	4.816	114.44	0.949
	Random	674	3895	25	1.298	3.288	5.779	4.715	116.23	0.939
$N25_{ut}$	Affected	802	4745	25	1.582	3.565	5.916	4.680	116.36	0.925
	Our	**362**	**1879**	**19**	**0.626**	**2.160**	**5.191**	**3.871**	**60.14**	**0.998**
	Naïve	802	4507	25	1.502	3.511	5.620	4.792	113.70	0.938
	Random	802	4739	25	1.580	3.564	5.909	4.684	116.34	0.926
$N30_{ut}$	Affected	911	5449	25	1.816	3.764	5.981	4.663	116.51	0.915
	Our	**440**	**2225**	**19**	**0.742**	**2.330**	**5.057**	**3.897**	**59.83**	**0.997**
	Naïve	911	5228	25	1.743	3.722	5.739	4.763	114.48	0.928
	Random	911	5446	25	1.815	3.764	5.978	4.665	116.49	0.916
$N35_{ut}$	Affected	1093	6341	25	2.114	3.949	5.801	4.626	115.02	0.902
	Our	**530**	**2567**	**20**	**0.856**	**2.450**	**4.843**	**3.829**	**59.63**	**0.997**
	Naïve	1093	6117	25	2.039	3.914	5.597	4.704	113.24	0.916
	Random	1093	6335	25	2.112	3.948	5.796	4.629	115.00	0.903
$N40_{ut}$	Affected	1313	7584	25	2.528	4.134	5.776	4.505	114.83	0.882
	Our	**649**	**3023**	**19**	**1.008**	**2.575**	**4.658**	**3.695**	**60.13**	**0.996**
	Naïve	1313	7330	25	2.443	4.100	5.583	4.569	112.86	0.895
	Random	1313	7582	25	2.527	4.134	5.775	4.506	114.82	0.882

TABLE 5: Results of the experiment with synthetic logs with incorrectly-ordered events inserted at unique trace level.

Log	Technique	# Affected Traces	Distance		Entire Log		Affected Traces	
			Cumulative	Max	Average Distance	Stand. Dev.	Average Distance	Stand. Dev.
BPI2014	Affected	36442	144957	63	3.110	3.445	3.978	3.425
	Our	**19838**	**77312**	**63**	**1.658**	**2.968**	**3.897**	**3.461**
	Naïve	35000	142072	63	3.048	3.482	4.059	3.471
	Random	34070	140199	63	3.008	3.505	4.115	3.501

TABLE 6: Results of the experiment with the real-life log.

artificial logs, and being 5.42 min for the BPI2014 log). This is due to the inherent complexity of estimating the probability density function (via kernel estimation) associated with the duration of each event.

6 Case Study

In collaboration with an Australian consultancy company we applied our technique to an event log from the meat & livestock domain. The process recorded in the log covers all stages livestock goes through from when cows are bred or purchased at the market, to when they are slaughtered and the meat is processed. Specifically, the event log contains data (a total of 18,192 events) originating from four different cattle farms, and records the processing of 3,032 cows over a timeframe of about two years. In this context, thus, each animal identifies a process case. Each cow is RFID-tagged, to be able to record the activity each cow participates in (e.g. breading, backgrounding, feeding), throughout the various stages of this process.

When analysing the event log, we realized it was affected by several timestamp errors. Specifically, the timestamp of each event did not capture the time when the event occurred (i.e. hours and minutes) but only the date. As discussed, this type of problem is very frequent in real-life event logs, and depends on the logging granularity of the logging system. In our particular case, this resulted in 2,748 cases where cows were slaughtered (activity "Processing") before leaving the feeding lot (activity "Feedlot - EXIT"), as can be observed in Figure 7a. This transition is not possible in reality, as confirmed by domain experts.

The application of our approach required no domain knowledge or additional processing of the log. Figure 7b shows the direct-follows graph of the log after repair, where the order of activities "Feedlot - EXIT" and "Processing" has been swapped. Our approach was able to repair all incorrect cases (requiring on average 66.5 ± 3.08 sec to repair the log). The resulting graph is less cluttered and so more readable. For example, the process cases that incorrectly transitioned from "Feedlot - PURCHASE" to "Processing" have now converged into the edge going from "Feedlot - PURCHASE" to "Feedlot - EXIT". This graph was validated by the business analyst involved in the case study, who confirmed the correct sequencing of activities.

7 Conclusion

We contributed an automated approach for correcting same-timestamp errors in event logs. These errors are quite common in real-life event logs, leading to quality issues in process mining. The approach consists of two techniques. The first one identifies the most likely sequence of events,

Log	Event reordering time [sec]				Timestamp estimation time [sec]				Total time [sec]			
	Avg	StDev	Min	Max	Avg	StDev	Min	Max	Avg	StDev	Min	Max
Event	7.77	6.05	1.61	20.21	345.50	454.60	110.89	1464.21	353.27	451.77	122.64	1465.82
Trace	3.50	1.85	0.86	5.89	245.81	36.54	198.74	298.46	249.30	34.71	204.63	299.31
UniqueTrace	3.60	1.60	1.41	6.24	249.08	37.54	196.12	308.79	252.68	35.97	202.36	310.19
BPI2014	2475.32	41.99	2421.84	2535.45	325.46	26.89	277.98	340.74	2800.78	24.08	2762.25	2825.19

TABLE 7: Time Performance for both artificial and real-life log experiments.

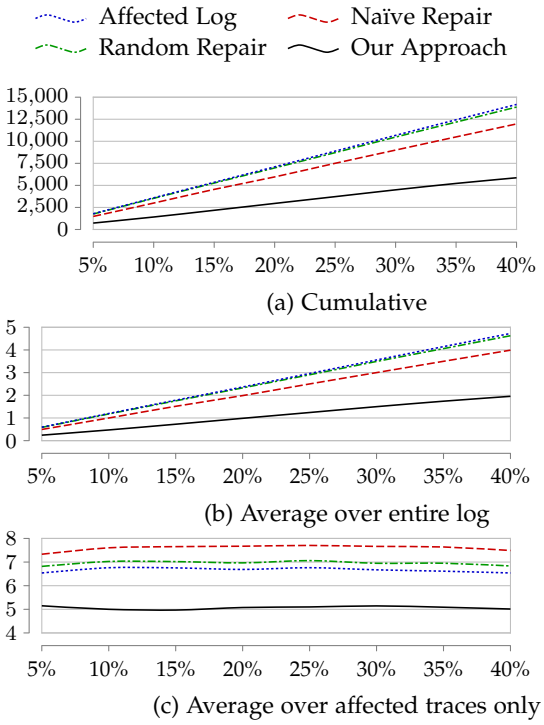


Fig. 5: Edit distance for the synthetic logs affected by incorrectly-ordered events at event level.

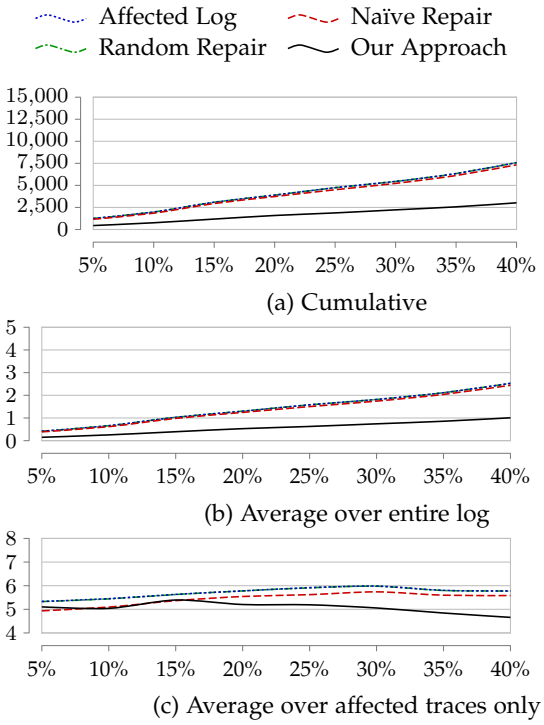


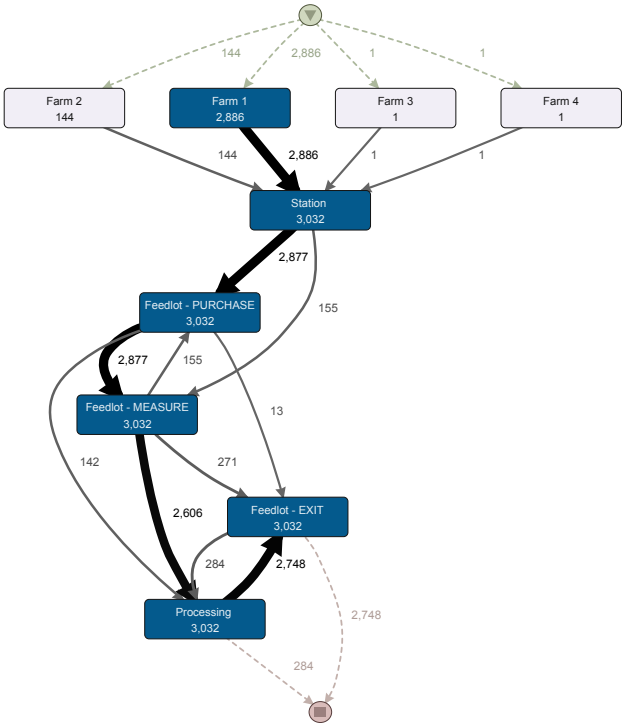
Fig. 6: Edit distance for the synthetic logs affected by incorrectly-ordered events at unique trace level.

for those events within a trace that have the same timestamp, and repairs the trace accordingly. Next, the second technique assigns a timestamp to each reordered event, based on a multimodal distribution of the duration of process activities, estimated using kernel density. Further, we discussed the inherent complexity of determining the most likely sequence of events and provided an ILP formulation of the problem.

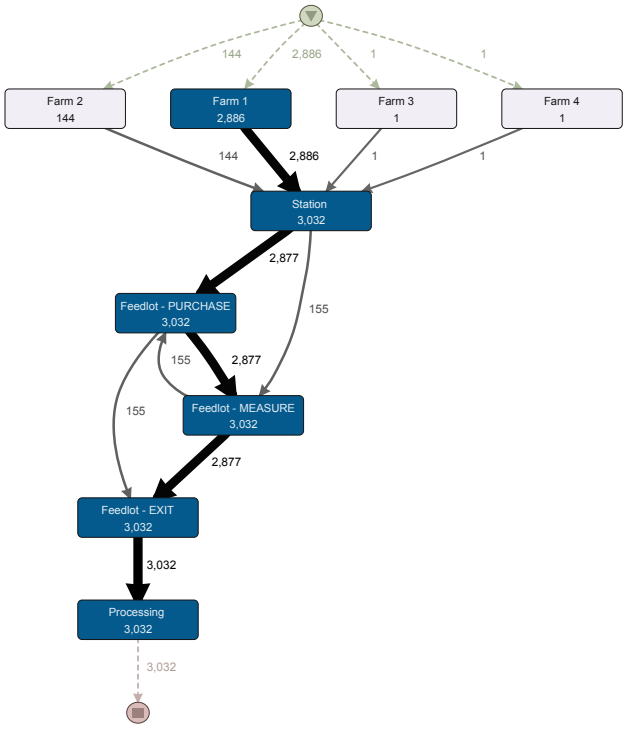
We implemented our approach in a software tool, and used the latter to extensively evaluate the approach using both synthetic and real-life logs. The experiments show that the approach reduces the amount of incorrect timestamps by at least 50%, both on synthetic as well as on real-life logs. In comparison, the two baseline approaches (random and naïve reordering) reduce the amount of incorrect timestamps by only 5%.

We complemented the evaluation with a case study conducted in the meat & livestock domain. The study shows the usefulness of our approach in practice: we find and repair all timestamp errors in this log, leading to the correct sequence of events, as confirmed by the business analyst involved in the study. Finally, time performance measurements show that the reordering of events scales well to large and complex logs, generally being able to reorder events in a few seconds. On the other hand, the estimation of timestamps for these events is more involving (in the order of minutes) due to the inherent complexity of this operation.

A possible avenue for future work is to fix the "inadvertent time travel" pattern described in [28]. This pattern captures the situation where a set of events are erroneously recorded with the wrong (yet different) timestamp by humans, because the corresponding activities are performed one shortly after the another. For example, this has been observed for activities performed just after midnight in healthcare logs. In these cases, the date is incorrectly recorded as that of the previous day, while the time is recorded correctly, resulting in erroneous timestamps. Another human error leading to this pattern is when the user inadvertently presses the keys adjacent to the intended ones, when entering an event's timestamp into a log. An idea is to adapt our approach by introducing a preliminary phase which, using our probabilistic model, identifies such cases and assigns them an equal timestamp. This will allows us to treat them as cases of the problem addressed within the context of this paper.



(a) Directly-follows graph of the log affected by timestamp errors.



(b) Directly-follows graph of the log after being repaired.

ACKNOWLEDGMENT

This research is partly funded by the ARC Discovery Project DP150103356.

REFERENCES

[1] Aria Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *Proceedings of the Fifteenth IEEE International Enterprise Distributed Object Computing Conference, EDOC 2011, Helsinki, Finland, August 29 - September 2, 2011*, pages 55–64. IEEE, 2011.

[2] Charu C. Aggarwal. *Outlier Analysis*. Springer, 2013.

[3] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F.M. Maggi, A. Marrella, M. Mecella, and A. Soo. Automated discovery of process models from event logs: Review and benchmark. Technical report, arXiv:1705.02288, 2017.

[4] Sabyasachi Basu and Martin Meckesheimer. Automatic outlier detection for time series: an application to sensor data. *Knowledge and Information Systems*, 11(2):137–154, 2006.

[5] Suratna Budalakoti, Ashok N. Srivastava, and Matthew E. Otey. Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 39(1):101–113, 2009.

[6] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2012.

[7] Raffaele Conforti, Marcello La Rosa, and Arthur H. M. ter Hofstede. Filtering out infrequent behavior from business process event logs. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):300–314, 2016.

[8] Kaustav Das, Jeff Schneider, and Daniel B. Neill. Anomaly pattern detection in categorical datasets. In *Proceedings of the Fourteenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'08)*, pages 169–176. ACM, 2008.

[9] Boudewijn F. van Dongen. BPI Challenge 2014: Activity log for incidents, 2014.

[10] Mohammadreza Fani Sani, Sebastiaan J. van Zelst, and Wil M. P. van der Aalst. Improving Process Discovery Results by Filtering Outliers using Conditional Behavioural Probabilities. In *Thirteenth International Workshop on Business Process Intelligence (BPI'17)*, 2017.

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

[12] Manish Gupta, Jing Gao, Charu C. Aggarwal, and Jiawei Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, 2014.

[13] Robert Gwadera, Mikhail J. Atallah, and Wojciech Szpankowski. Reliable detection of episodes in event sequences. *Knowledge and Information Systems*, 7(4):415–437, 2005.

[14] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[15] S.S. Jambhorkar and V.S. Jondhale. *Data Mining Technique: Fundamental Concept And Statistical Analysis*. Horizon Books, 2015.

[16] Eamonn Keogh, Jessica Lin, Sang-Hee Lee, and Helga van Herle. Finding the most unusual time series subsequence: algorithms and applications. *Knowledge and Information Systems*, 11(1):1–27, 2006.

[17] Eamonn Keogh, Stefano Lonardi, and Bill'Yuan-chi Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of the Eighth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'02)*, pages 550–556, 2002.

[18] Marcello La Rosa, Hajo A. Reijers, Wil M. P. van der Aalst, Remco M. Dijkman, Jan Mendling, Marlon Dumas, and Luciano García-Bañuelos. APROMORE: an advanced process model repository. *Expert Systems with Applications*, 38(6):7029–7040, 2011.

[19] Terran Lane and Carla E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *Proceedings of the AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, 1997.

[20] Terran Lane and Carla E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, 1999.

[21] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[22] Moshe Lewenstein and Maxim Sviridenko. Approximating asymmetric maximum tsp. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'03, pages 646–654. Society for Industrial and Applied Mathematics, 2003.

[23] Clair E. Miller, Albert W. Tucker, and Richard A. Zemlin. Integer

programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.
[24] Shan Muthukrishnan, Rahul Shah, and Jeffrey S. Vitter. Mining deviants in time series data streams. In *Proceedings of the Sixteenth IEEE International Conference on Scientific and Statistical Database Management SSDMN'04*, pages 41–50. IEEE, 2004.
[25] D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, Los Altos, California, 1999.
[26] Bernard W. Silverman. Using kernel density estimates to investigate multimodality. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 97–99, 1981.
[27] Pei Sun, Sanjay Chawla, and Bavani Arunasalam. Mining for outliers in sequential databases. In *Proceedings of the 2006 SIAM International Conference on Data Mining (ICDM'06)*, pages 94–105. Society for Industrial and Applied Mathematics, 2006.
[28] S. Suriadi, R. Andrews, A.H.M. ter Hofstede, and M.T. Wynn. Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. *Information Systems*, 64, 2017.
[29] Suriadi Suriadi, Robert Andrews, Arthur H. M. ter Hofstede, and Moe T. Wynn. Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. *Information Systems*, 64:132–150, 2017.
[30] W.M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016.
[31] Eric Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Prom 6: The process mining toolkit. In *Proceedings of the Business Process Management 2010 Demonstration Track (BPM Demo'10)*, volume 615 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.
[32] Jianmin Wang, Shaoxu Song, Xuemin Lin, Xiaochen Zhu, and Jian Pei. Cleaning structured event logs: A graph repair approach. In *Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE'15)*, pages 30–41. IEEE, 2015.
[33] Dragomir Yankov, Eamonn Keogh, and Umaa Rebbapragada. Disk aware discord discovery: finding unusual time series in terabyte sized datasets. *Knowledge and Information Systems*, 17(2):241–262, 2008.
[34] S.J. Zelst, M.F. Sani, A. Ostovar, R. Conforti, and M. La Rosa. Filtering Spurious Events from Event Streams of Business Processes. In *Proceedings of the International Conference on Advanced Information Systems Engineering*, LNCS. Springer, 2018.
[35] Xiaoqiang Zhang, Pingzhi Fan, and Zhongliang Zhu. A new anomaly detection method based on hierarchical hmm. In *Proceedings of the Fourth IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'03)*, pages 249–252. IEEE, 2003.

Raffaele Conforti is a Lecturer at the University of Melbourne, Australia. He conducts research on process mining and automation, with a focus on automated process discovery, quality improvement of process event logs and process-risk management.

Marcello La Rosa is a Professor of Information Systems at the University of Melbourne, Australia. His research interests include process mining, consolidation and automation. He leads the Apromore Initiative (www.apromore.org), a cross-university collaboration for the development of an advanced process analytics platform, and co-authored the textbook "Fundamentals of Business Process Management" (Springer, 2nd edition).

Arthur H.M. ter Hofstede is a Professor and Head of the School of Information Systems in the Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia. His research interests are in the areas of process automation and process mining, where he has contributed numerous papers.

University Library

A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Conforti, R; La Rosa, M; ter Hofstede, A

Title:
Timestamp Repair for Business Process Event Logs

Date:
2018-04-05

Citation:
Conforti, R; La Rosa, M; ter Hofstede, A, Timestamp Repair for Business Process Event Logs, 2018

Persistent Link:
http://hdl.handle.net/11343/209011

File Description:
Submitted version