Pattern-based Reconstruction of Anomalous Traces in Business Process Event Logs

Jonghyeon Ko[1], Marco Comuzzi[2]

[1] *Institute for the 4th Industrial Revolution and* [2] *Department of Industrial Engineering*
Ulsan National Institute of Science and Technology (UNIST)
Ulsan, Republic of Korea

Abstract

Event log anomaly detection and log repairing concern the identification of anomalous traces in an event log and the reconstruction of a correct trace for the anomalous ones, respectively. Anomalies in real-world event logs often appear according to specific patterns, since they are generated by a limited number of root causes, such as poor resource behaviour or system malfunctioning. This paper proposes PBAR (Pattern-Based Anomaly Reconstruction), a semi-supervised pattern-based anomaly detection and log repairing approach that exploits the pattern-based nature of trace-level anomalies in event logs. PBAR captures in a set of ad-hoc graphs the behaviour of clean traces in a log and uses these to identify anomalous traces, the specific anomaly pattern that applies to them, and then reconstruct the correct trace. The proposed approach is evaluated using artificial and real event logs against the traditional trace alignment in conformance checking method, the edit distance-based alignment method, and an unsupervised method based on deep learning.

Keywords

Business Process Event Log, Event Log Quality, Anomaly Pattern, Anomaly Reconstruction

1. Introduction

Process mining [1, 2] is a research field at the crossroads of process science and data science that focuses on improving business processes through the analysis of the data logged during their execution in so-called *event logs*. To enable basic process mining analysis, event logs must contain, for each task executed in a business process, an id of the process case to which it belongs and an activity label. Events must also be ordered in time.

In practical situations, event logs are prone to errors [3, 4, 5, 6], which are caused by often unavoidable or unexpected system failures or human errors. For instance, a doctor in a hospital may forget to log the start time of a consultation with a patient, whereas a number of events in a log may be missing because an application used in a process was unavailable due to a network failure for a certain period. Low quality event logs can crucially disrupt process mining-based analyses.

The quality of data is a general concern in data science, where the statement *"garbage in, garbage out"* is often adopted to highlight that low input data quality leads to low quality of the

International Workshop on Computational Intelligence for Process Mining (CI4PM)
whd1gus2@unist.ac.kr (J. Ko); mcomuzzi@unist.ac.kr (M. Comuzzi)
0000-0002-8322-8056 (J. Ko); 0000-0002-6944-4705 (M. Comuzzi)

results of the analysis [7, 8]. In this context, *anomaly detection* on input data is seen as a key technique to improve their quality and, therefore, support higher quality analysis [5].

The follow-up of anomaly detection is *log repairing*, that is, the problem of reconstructing a correct trace for the ones that have been identified as anomalous. Besides enabling the correct reconstruction of specific anomalous traces, an effective method to repair a log may increase the number of observations (traces) available for other process mining use cases, which can be a particularly critical issue when a high number of traces in a log are anomalous because of systematic logging errors.

Existing research on log repairing often does not consider the crucial fact that anomalies in event logs are not completely random, but they stem from specific anomaly *patterns* [6, 9, 10]. For instance, sloppy or poorly trained human resources may forget to log some events, log multiple times the same event, log a different task in place of the one actually executed, or log a wrong timestamp for an event, which will result in an event moving before or ahead its correct intended position in a trace [6].

In this paper, we propose PBAR (Pattern-Based Anomaly Reconstruction), a semi-supervised, pattern-based, trace-level event log anomaly detection and log repairing approach. PBAR is a *pattern-based* approach, because we assume that the type of patterns that determine anomalies in a log are known a priori, and it is *semi-supervised*, because we assume that a set of clean traces is available in an event log.

First, PBAR uses the clean traces to construct a set of directed graphs that capture the sequential relations among events in a log in clean traces. Then, for each trace with unknown label (normal v. anomalous), we propose an algorithm that, through a custom replay of a trace on the directed graphs created at the previous step, yields an anomaly detection matrix. The values in this matrix exhibit specific patterns depending on the anomaly pattern that affected a trace. Since the reconstruction of a trace relies on the identification of the correct anomaly pattern affecting a trace, the proposed approach is *interpretable* by the design, i.e., not only does it return a reconstructed trace, but it also *explains* the reason why a trace has been identified as anomalous and has been reconstructed in that particular way.

We have evaluated PBAR on artificial and real-life event logs publicly available and using anomaly patterns commonly considered by other research works in this field. The performance of the proposed model, both in term of reconstruction accuracy and run time, has been compared against baseline reconstruction methods of different types, i.e., traditional and edit distance-based trace alignment (model-aware), and DeepAlign [11], a model-agnostic approach that uses deep learning.

The remainder of the paper is organised as follows. Section 2 presents the related work. Sections 3 and 4 define the problem of event log anomaly detection and reconstruction and the PBAR approach to solve it, respectively. PBAR is evaluated in Section 5, whereas conclusions are finally drawn in Section 6.

2. Related work

Model-aware trace-level anomaly detection and repairing exploit a given reference process model, such as a workflow net or a BPMN model. Since process models capture the control flow

of a process, these approaches mainly focus on reconstructing the correct order of events in traces. Wang et al. [12] have introduced an approach that, using a workflow net model, exploits a branch algorithm to yield multiple choices for reconstructing process execution flows with missing events. Song et al. [13] have proposed a heuristic log repair strategy based on a Petri net decomposition of the process model, which allows to identify the minimum recovery of an incomplete sequence in one of several independent sub-processes. Rogge Solti et al. [14] have introduced an alignment approach based on Petri nets to repair missing events in event logs. Dixit et al. [15] have approached the handling of event log noise using temporal ordering of events and repairing the noise through an alignment approach. In summary, model-aware approaches repair missing events or infrequent traces in an event log by finding a complete trace with minimum cost (distance) from a reference model, often exploiting compliance checking techniques. In other words, model-aware approaches are helpful to reconstruct traces in an event log under well-defined process specifications.

As far as model-agnostic approaches are concerned, Sani et al. [8] have introduced a probabilistic method to define frequent context behaviour patterns in an event log, which then are used to repair infrequent behavior patterns with more frequent patterns. Since the patterns are defined using a threshold on the relative measure of behavioural context frequency, this approach implicitly assumes that data with infrequent patterns are anomalous. Xu and Liu [16] have introduced a distance-based method that uses the trace clustering of log profiles. The log profiles cannot be applied to different anomaly patterns, such as moving events in a trace, since they do not consider the order in control-flows. Similarly to model-aware approaches, most model-agnostic approaches have focused on the issue of missing or infrequent events in a log. The pattern-based approach to event log anomaly modelling, which we consider in this work, has emerged only recently with the work of Nolle et al. [11]. Such an approach is also aligned with recent work on event log data quality [10, 9, 5, 6], which has demonstrated that often in real world situations most anomalies in an event log belong to few specific patterns determined by a few specific root causes, such as human resource error or system malfunctioning.

3. Problem definition

Given a set X, we use X^* and $\mathbb{B}(X)$ to denote the set of all finite sequences over it and the set of all multisets over it, respectively. Let A be a set of activity labels. A trace $\sigma_j \in A^*$ is the sequence of the I activities $a_{1,j}, \ldots, a_{i,j}, \ldots a_{I,j}$ executed in the j-th process case. An event log $E \in \mathbb{B}(A^*)$ is a multiset of traces. For convenience, we consider an augmented version of traces with fictitious start and end events at position $i = 0$ and $i = I + 1$, respectively, i.e., $\sigma_j = [start, a_{1,j}, \ldots, a_{I,j}, end]$.

Similarly to recent literature on event log anomaly detection [5, 10, 9], we consider 5 trace-level anomaly patterns normally linked to poor resource behaviour as a root cause: *Skip*, *Insert*, *Rework*, *Replace*, and *Move* one or more events in a trace. These are non-trivial anomalies that cannot be easily identified by frequency-based filters typically available in commercial process mining tools. The 5 patterns are exemplified in Figure 1 and defined next:

- $Skip$: given a trace σ_j of the length I, one activity $a_{i,j} \in \sigma_j$ is deleted from σ_j.

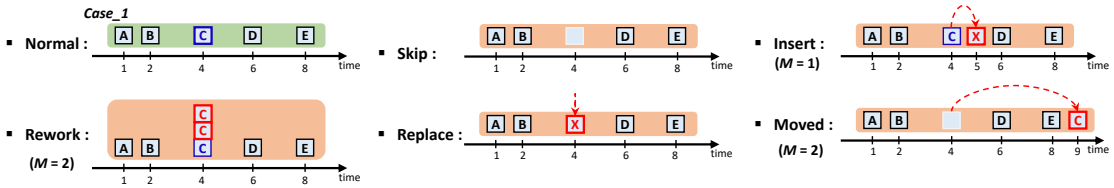

Figure 1: Trace-level anomaly patterns considered in this work

- $Insert(M)$: given a trace σ_j of the length I and a positive integer M, an integer value m is first randomly chosen in $[1, M]$. Then, m activities $a_{k,j} \in \sigma_j$, with $k \in [1, m]$, are randomly chosen. Finally, m activities are generated using random activity labels in σ_j after the activity $a_{i,j} \in \sigma_j$.
- $Rework(M)$: given a trace σ of the length I and positive integers M, an integer value m is randomly chosen in $[1, M]$. Then, one activity $a_{i,j} \in \sigma_j$ is repeated m times after its occurrence.
- $Replace$: given a trace σ_j of the length I, one activity $a_{i,j} \in \sigma_j$ is changed to a different activity label.
- $Move(M)$: given a trace σ of the length I and a positive integer M, an integer value $m \neq 0$ is randomly chosen in $[-M, M]$. Then, one activity $a_{i,j} \in \sigma_j$ is moved of m events from its current location.

In a semi-supervised anomaly detection approach, a set of clean observations is available. Therefore, we define a labelling function $L : A^* \rightarrow \{n, a\}$ that associates a trace $\sigma \in A^*$ to its label: normal (n) or anomalous (a). We assume that an event log E is partitioned into a (normally small) log containing only clean traces E_c, i.e., $\forall \sigma \in E_c$, $L(\sigma) = n$, and the set of the remaining traces $E_o = E \setminus E_c$. While the label of the traces in E_c is known, the label of the traces in E_o is a ground truth generally unknown.

The reconstruction accuracy entails a strict notion of correctness: if the reconstructed trace matches exactly the original one, then the accuracy is 1, otherwise it is 0, i.e., $match(\sigma_1, \sigma_2) \rightarrow \{1, 0\}$. A more grey-shading notion of accuracy can be introduced by considering the distance between traces. Following similar work in event log anomaly detection [11], in this work we consider a distance $d : A^* \times A^* \longrightarrow [0, 1]$ between two traces σ_1 and σ_2 defined using the Levensthein string distance $lev(\cdot)$ [17]. Specifically, given a bijective mapping $n : A \rightarrow S$ between the activity labels A and an alphabet S, and a function $N : A^* \rightarrow S^*$ translating a trace σ into a string defined over S, $d(\sigma_1, \sigma_2) = lev(N(\sigma_1), N(\sigma_2))$.

Now, a trace repair function $R : A^* \rightarrow A^*$ can be defined to associate a trace σ to a reconstructed trace $R(\sigma)$. Given a trace σ with $\hat{L} = a$, the accuracy of the trace reconstruction is assessed by using the two notions of accuracy defined above: $match(\sigma, R(\sigma))$ and $d(\sigma, R(\sigma))$. Specifically, the lower the distance, the more accurate the reconstruction for the distance $d(\sigma, R(\sigma))$.

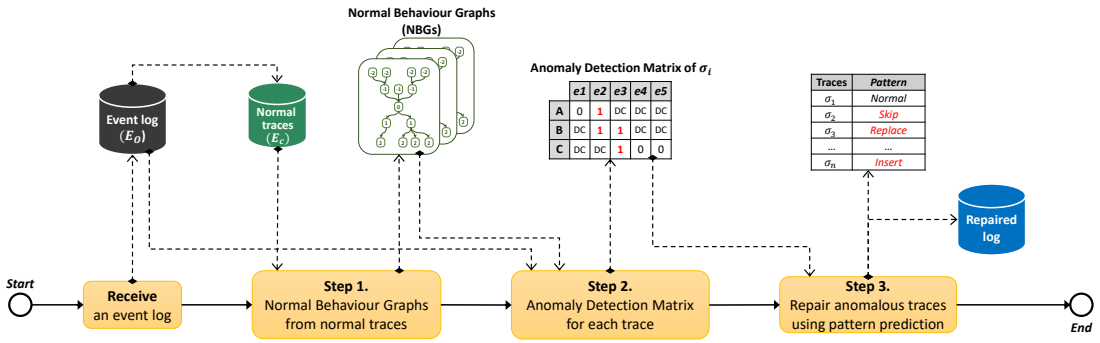

Figure 2: Framework of PBAR

Algorithm 1: [Step 1.] Processing traces to create T_k

Inputs: E_c, a_k
Output: T_k

$T_k \leftarrow \langle V_k \leftarrow \emptyset, E_k \leftarrow \emptyset \rangle$ ▷ empty graph
$l \leftarrow 0$ ▷ level
for all $\sigma_j \in E_c : a_k \in \sigma_j$
$p \leftarrow i : a_{i,j} \in \sigma_j = a_k$
  while $a_{p,j} \neq start$ ▷ process prefix
    $V_k \leftarrow V_k \cup (a_p, l) \cup (a_{p-1}, l-1)$
    $E_k \leftarrow E_k \cup ((a_{p-1}, l-1), (a_p, l))$
    $p \leftarrow p - 1$
    $l \leftarrow l - 1$
  $l \leftarrow 0$
  $p \leftarrow i : a_{i,j} \in \sigma_j = a_k$
  while $a_{p,j} \neq end$ ▷ process suffix
    $V_k \leftarrow V_k \cup (a_p, l) \cup (a_{p+1}, l+1)$
    $E_k \leftarrow E_k \cup ((a_p, l), (a_{p+1}, l+1))$
    $p \leftarrow p + 1$
    $l \leftarrow l + 1$
return T_k

Algorithm 2: [Step 2.] Compiling the content of D^j

Inputs: σ_j, $T \leftarrow \cup_k T_k$
Output: D_j

$\forall p, q\ d_{p,q} \in D^j \leftarrow 0$ ▷ Initialise D^j
$p \leftarrow 0$ ▷ a_p current activity in σ_j
while $p \leq I + 1$ **do**
  $l \leftarrow 0$
  $q \leftarrow p$
  while $q \neq 0$ ▷ replay prefix
    $e \leftarrow ((a_{q-1}, l-1), (a_q, l))$
    if $e \in T_{a_p}$
      $m_{p,q} \leftarrow 0$ ▷ replay correct
    else
      $m_{p,q} \leftarrow 1$
      $\forall x : 0 \leq x < q,\ m_{p,x} \leftarrow DC$
    $l \leftarrow l - 1$
    $q \leftarrow q - 1$
  $l \leftarrow 0$
  $q \leftarrow p$
  while $q \neq I + 1$ ▷ replay suffix
    $e \leftarrow ((a_q, l), (a_{q+1}, l+1))$
    if $e \in T_{a_p}$
      $m_{p,q} \leftarrow 0$ ▷ replay correct
    else
      $m_{p,q} \leftarrow 1$
      $\forall x : q < x \leq I + 1,\ m_{p,x} \leftarrow DC$
    $l \leftarrow l - 1$
    $q \leftarrow q - 1$
  $p \leftarrow p + 1$ ▷ process next activity
return D^j

4. Framework

Figure 2 depicts the overall framework of PBAR. The input is an event log, from which a set E_c of normal traces is separated from a set E_o of traces with unknown label (we only assume that the label of E_o is known when evaluating the proposed approach). Then, PBAR comprises 3 phases: (i) creating a set of directed graphs that capture the sequential relations among activities in clean traces (the Normal Behaviour Graphs - NBGs), (ii) calculating, for each trace in E_o,

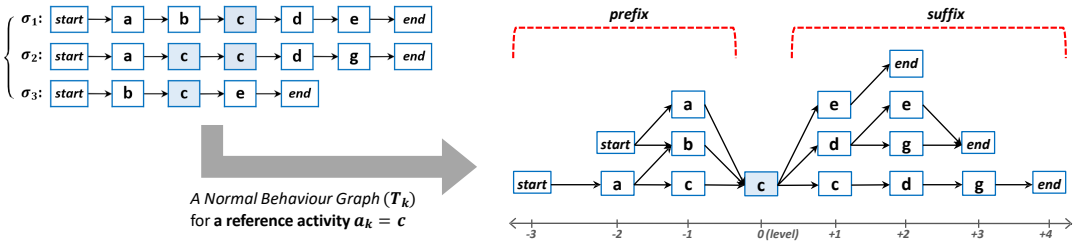

Figure 3: Compiling the anomaly detection matrix: example

an anomaly detection matrix by replaying traces on the NBGs obtained at the Step (i), and (iii) detect anomalous traces and reconstruct them through the pattern-based analysis of the anomaly detection matrix.

4.1. Normal Behaviour Graphs

As a semi-supervised learning method to obtain a reference model representing normal behaviours, PBAR creates a normal behaviour graph $T_k = \langle V_k, E_k \rangle$ for each activity $a_k \in A$ using the traces in E_c, where V_k is a set of nodes and E_k is a set of edges, with $E_k \subseteq V_k \times V_k$. The nodes in T_k are identified by an activity label and a level $l \in \mathbb{Z}$, i.e., $V_k \subseteq A \times \mathbb{Z}$.

Given a trace σ_j, the prefix function $prefix : A^* \times A \longrightarrow A^*$ and the suffix function $suffix : A^* \times A \longrightarrow A^*$ return the events in σ_j before and after an activity $a_p \in \sigma_j$, respectively: $prefix(\sigma_j, a_p) = \{start, a_{1,j}, \ldots, a_{p-1,j}\}$ and $suffix(\sigma_j, a_p) = \{a_{p+1,j}, \ldots, a_{I,j}, end\}$.

The graph T_k for an activity a_k is created by processing all the traces in E_c that contain the activity a_k as shown in Alg. 1. A trace σ_j that contains a_k is split into a prefix and a suffix where a_k occurs, which are processed separately. The prefix is processed from a_k backward to the start of σ_j. At each iteration, an edge $e \in E_k$, i.e., a pair of consecutive events $((a_{i-1}, l-1), (a_i, l))$ at a certain level l is created if does not exist already. The level l is set to 0 initially and it decreases as the trace σ_j is scanned backward. The processing of suffix part entails a similar logic, with the only difference that a suffix is processed from a_k forward until the end of σ_j and that, therefore, the value of the level l is increased as the suffix of σ_j is scanned forward.

Figure 3 shows an example of how T_c is created for the activity $a_k = c$, given 3 normal traces, i.e., $E_c = \{\sigma_1, \sigma_2, \sigma_3\}$, with $\sigma_1 = [start, a, b, c, d, e, end]$, $\sigma_2 = [start, a, c, c, d, g, end]$, and $\sigma_3 = [start, b, c, e, end]$.

4.2. Anomaly Detection Matrix

The anomaly detection analysis relies on a custom *replay* of a trace in E_o on the NBGs T_k. Given a set of NBGs obtained at the Step 1 of the PBAR framework, such a replay of a trace σ_j yields a matrix D^j. In this matrix, rows are identified by the ordered activities in a trace, whereas columns are identfied by the directly-follows relations in the trace: given a trace $\sigma_j = \{start, a_{1,j}, \ldots, a_{I,j}, end\}$, the matrix D^j has $I + 2$ rows, and $I + 1$ columns, identified by the directly follows relations in a_j, e.g., $start \rightarrow a_{1,j}$ or $a_{1,j} \rightarrow a_{2,j}$. Each element $d^j_{n,m} \in D^j$

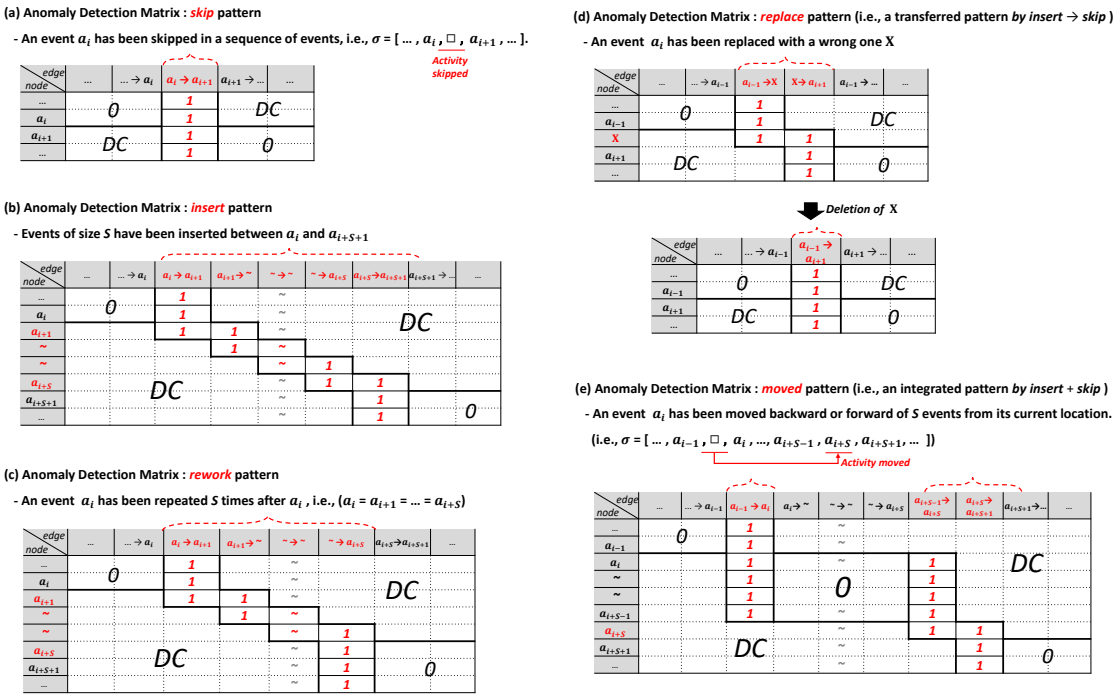

Figure 4: Anomaly detection matrix by different anomaly patterns

of this matrix assumes one of three values [0, 1 or DC (do not care)] as a result of the custom replay of σ_j, which is presented in Alg. 2.

Similarly to creating the NBGs, the replay of σ_j is split between replaying the prefix (lines 6-14) and the suffix (17-25). In either case, when replaying an activity $a_{i,j} \in \sigma_j$, the NBG $T_{a_{i,j}}$ is considered. Starting from the level 0 in $T_{a_{i,j}}$, the prefix of σ_j is replayed backward, aiming to reach the start of σ_j, whereas the suffix is replayed forward, aiming to reach the end of σ_j. When scanning the prefix [suffix], if an edge $((a_{i-1}, l), (a_i, l))$ [$((a_i, l), (a_{i+1}, l))$] exists in $T_{a_{i,j}}$, then the corresponding value in the matrix D^j is set to 0. Otherwise, the corresponding value in the matrix is set to 1. Since, after a value 1 is set in D^j, a trace can no longer be replayed on $T_{a_{i,j}}$, then all the preceding [subsequent] values to 1 in the same row are set to DC's.

4.3. Anomaly Reconstruction

In this section, we show how the anomaly detection matrix is used for detecting anomalies and reconstructing them. Note that, as introduced before, we assume that there are 5 possible trace-level anomaly patterns, i.e., $skip$, $insert$, $rework$, $replace$, and $move$, in event logs, and that an individual trace is affected by at most one anomaly pattern.

Detecting anomalies is straightforward: a trace is anomalous if its anomaly detection matrix D^j contains at least one element $d^j_{n,m} = 1$, that is, if at least one activity in a trace could not be replayed correctly according to the procedure described in Alg. 2. By design, the values $d^j_{n,m} = 1$ in the anomaly detection D^j follow specific patterns, which is the property exploited in the

trace reconstruction phase. Next we present in detail how each anomaly pattern is detected and reconstructed.

Skip pattern. The skip pattern [see Fig. 4(a)] is clearly identifiable in an anomaly detection matrix when a column has only values equal to 1. Such a column identifies that one activity between a_i and a_{i+1} has been anomalously skipped. Regarding the reconstruction of this type of traces, following the approach proposed by Sani et al. [8], the reconstruction occurs by inserting into σ_j the most frequent activity at position between i and $i+1$.

Note that, in general, the anomaly detection matrix is recalculated for every reconstructed trace: if this matrix does not signal any anomaly, then the next trace can be considered, i.e., the reconstruction has been successful; if this matrix signals an anomaly, i.e., it contains a value equal to 1, then the pattern-based analysis is run again.

Insert pattern. Figure 4(b) shows the pattern in the anomaly detection matrix identifying the insertion of S unexpected events in a trace between the events a_i and a_{i+S+1}. For the reconstruction, the S events inserted are deleted from the trace.

Rework pattern. The rework pattern is a special case of the insert one in which all the events inserted have the same activity label (see Figure 4(c)). As such, the detection and reconstruction of this pattern is already captured by what described above for the insert pattern.

Replace pattern. From the standpoint of anomaly injection, replacing an activity in a trace is equivalent to skipping it, i.e., deleting an existing activity, and inserting a different one in its place. The anomaly detection and reconstruction heuristic inverts this logic, by first identifying whether an anomalous activity has been inserted in a trace and then checking whether it should be replaced (or simply deleted). Hence [see Figure 4(d)], detecting the replace pattern starts by detecting the insert pattern (with $S = 1$) in the anomaly detection matrix. Once the inserted activity has been deleted, the anomaly detection matrix is recalculated, highlighting the skip pattern at position a_1. Then, following the reconstruction of the skip pattern, a correct activity to be inserted at position a_i is determined.

Move pattern. From an anomaly injection standpoint, the move pattern can be seen as the sequential application of the skip (i.e., an activity is removed from its normal place) and insert (i.e., the same activity is inserted at a different place) patterns. As such, this pattern is identifiable in the anomaly detection matrix through a combination of the insert and skip patterns described above [Figure 4(e)].

More details about exceptional cases in the anomaly reconstruction process are discussed in an online github repository that accompanies this paper. [1]

5. Evaluation

5.1. Experimental Datasets

For the evaluation of the proposed approach, we have considered the 5 artificial logs used in [5] and 2 publicly available real-life event logs. Regarding real-life event logs, we consider the *Hospital Billing* event log, which collects events from a billing of medical services process in a

[1] https://github.com/paai-lab/Pattern-based-Anomaly-Reconstruction-2022

regional hospital and the *Road Traffic* event log, which collects events about a road traffic fine management process at a local police authority in Italy.

The anomaly injection process is supported by AIR-BAGEL [6], a tool developed by the authors that simulates anomalies in event logs based on two different types of root causes, i.e., sub-optimal resource behaviour and system malfunctioning. More details about the event logs considered, the anomalies injected in them, together with details of additional experimental results not discussed in the remainder of this section are available in the online companion github repository.

5.2. Evaluation Metrics and baseline models

As defined in Section 3, we evaluate three aspects of PBAR, which answer two different research questions:

- "How accurate is PBAR, given an anomalous trace, at reconstructing the correct one?" (Anomalous trace reconstruction accuracy).
- "How long does it take to execute PBAR on all the traces of an event log?" (Run time).

Regarding baseline models, we consider three approaches: (i) the traditional alignment method in conformance checking (Alignment.TR) [18], (ii) the edit distance-based alignment method (Alignment.ED) proposed by Schuster et al. [19], and (iii) DeepAlign [11]. For Alignment.TR, we first use the clean traces in an event log to discover a process model using the inductive miner [20] implementation in the Python module PM4Py [21]; then we align – using the alignment implementation available in PM4Py – the discovered process model and the anomalous traces, considering the aligned traces as the reconstructed ones. Both process discovery and alignment are run using the default parameter values. Since we use only clean traces for process discovery, the parameter of noise threshold in the inductive miner is set to 0. For Alignment.ED, we consider the implementation of the edit distance-based version of alignment based on the Levenshtein distance [17] available in the Python module PM4Py. In DeepAlign, sequence alignments are calculated exploiting a predictive model of next event in a running trace based on a Recurrent Neural Network (RNN). We use the same hyperparameter values adopted in the original paper [11]: $epochs = 50$ and $batch\_size = 100$, obtained from the Adam optimizer with standard parameters, $max\_iterations = 10$, beam size $K = 5$ and maximum deletion size $N = 3$. Note that the traditional alignment methods are model-aware approaches to anomaly reconstruction, whereas DeepAlign is a model-agnostic one (i.e., it does not discover a process model explicitly).

5.3. Experimental Results

While in this section we concentrate on the results that answer the research questions, results regarding the accuracy of PBAR broken down by type of anomaly pattern are available in the companion github repository.

Table 1 compares the reconstruction accuracy, calculated using both the classification accuracy and the average distance, and the run time of PBAR and the baselines. Note that, since DeepAlign is an unsupervised approach, which does not require a case label in the input dataset, for it we

Table 1
Trace reconstruction accuracy, error and run time. For reconstruction accuracy, the higher the better, for distance accuracy, the lower the better. The best performance for each metric and event log is highlighted in bold.

Performance metric	Method	Small	Medium	Large	Huge	Wide	Hospital Billing	Road Traffic	Average
ACC	**PBAR**	0.991	**0.936**	**0.957**	**0.977**	**0.947**	0.540	0.554	**0.787**
	Alignment.TR	**0.996**	0.922	0.941	0.975	0.944	0.168	0.183	0.597
	Alignment.ED	0.977	0.919	0.938	0.950	0.934	0.303	0.523	0.717
	DeepAlign (normal)	1.000	1.000	1.000	1.000	1.000	0.573	0.842	0.870
	DeepAlign (anomaly)	0.921	0.773	0.883	0.905	0.842	**0.589**	**0.693**	0.771
Error: Levenshtein Distance	*Before reconstruction*	*2.410*	*2.141*	*1.511*	*1.647*	*2.536*	*3.674*	*4.398*	*2.735*
	PBAR	0.021	0.083	0.062	0.037	0.075	**0.812**	**0.635**	**0.340**
	Alignment.TR	**0.008**	**0.078**	**0.060**	**0.025**	**0.058**	1.241	1.231	0.580
	Alignment.ED	0.054	0.087	0.063	0.070	0.078	1.178	0.717	0.432
	DeepAlign (normal)	0.000	0.000	0.000	0.000	0.000	0.669	0.317	0.220
	DeepAlign (anomaly)	0.261	0.397	0.121	0.141	0.242	0.933	0.825	0.496
Run time (minute)	PBAR	1.74	1.23	3.21	1.77	1.16	0.51	14.62	4.17
	Alignment.TR	0.04	0.04	0.12	0.08	0.03	0.03	0.43	0.13
	Alignment.ED	**0.00**	**0.00**	**0.02**	**0.00**	**0.00**	**0.00**	**0.02**	**0.00**
	DeepAlign	5.64	3.83	7.62	6.84	2.91	113.02	453.34	107.517

show separately the reconstruction accuracy on normal traces and anomalous traces. That is, an unsupervised method may generate errors even on normal traces. Otherwise, since both the proposed approach and the alignment baselines are semi-supervised, i.e., they require a set of clean labelled traces in input, the reconstruction accuracy of both approaches on normal traces is obviously perfect. Finally, note that, for the distance accuracy, we also report the average distance between normal and reconstructed traces before the reconstruction as a reference.

When compared with the baselines, PBAR shows the highest classification accuracy on most artificial logs, whereas DeepAlign emerges as the best performer in classification accuracy on real-world event logs. Note, however, that, because of its unsupervised nature, DeepAlign does not have perfect reconstruction accuracy on normal traces. This phenomenon is not negligible in the case of some real-world event logs. For instance, DeepAlign classifies incorrectly 42.7% of the normal traces in the Hospital Billing event log. Regarding the distance accuracy, PBAR is the top performer on the real world event logs, while achieving a performance comparable with the baselines on the artificial logs.

Finally, regarding run time, both PBAR and the alignment baselines execute more quickly than DeepAlign, particularly with real-world event logs. Note that the implementations of the alignment baselines in PM4Py are optimised to exploit parallel processing in multi-core architectures, which justifies the short run time (always less than a minute, down to few seconds for Alignment.ED). Our implementation of PBAR is not optimised for multi-core architectures and, therefore, it is characterised by higher run time than the alignment baselines, but still lower than the one of DeepAlign.

To conclude, since the alignment method works effectively with logs generated from well-structured processes [22], the experimental results show that the alignment baselines are often

not effective at reconstructing correctly the traces generated by real world event logs, which are generated by less structured and more flexible business processes. As a consequence, while the alignment baselines shows high reconstruction accuracy on the artificial logs, their overall performance is relatively lower than the one of PBAR or DeepAlign because of the low performance on real logs. The DeepAlign method shows overall fine performance while, however, it may mis-classify normal traces as anomalous, particularly in real world event logs.

6. Conclusions

We have presented PBAR, a pattern-based semi-supervised approach to identify trace-level anomalies in event logs and reconstruct correct traces. The proposed method uses the normal traces in a log to generate a set of activity-specific directed graphs capturing the correct process behaviour and then exploits these graphs to classify and reconstruct the unlabelled traces. We have evaluated the method against the traditional alignment method in conformance checking, the edit distance-based alignment method, and DeepAlign, which is a RNN-based unsupervised method, using both artificial and real-world event logs. As far as trace reconstruction is concerned, PBAR achieves the best performance on artificial logs. While PBAR is outscored by DeepAlign regarding the reconstruction classification accuracy, it achieves the highest distance-based average reconstruction accuracy. One of the main features of PBAR is that is interpretable, that is, the information generated during the anomaly detection and reconstruction process, i.e., the anomaly detection matrices, provide additional insights for decision makers regarding which anomaly pattern has affected a trace. To enhance the practical applicability of anomaly detection methods in practice, future extensions should also tackle the issue of handling multiple anomalies pattern applied to a single trace. This is in fact a situation that has not been considered by any of the approaches in the literature that we have reviewed, including the ones considered in the experimental evaluation of this paper.

References

[1] L. Maruster, A. T. Weijters, W. W. Van der Aalst, A. van den Bosch, Process mining: Discovering direct successors in process logs, in: International Conference on Discovery Science, Springer, 2002, pp. 364–373.

[2] W. Van Der Aalst, Process mining, Communications of the ACM 55 (2012) 76–83.

[3] S. Suriadi, R. Andrews, A. H. ter Hofstede, M. T. Wynn, Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs, Information Systems 64 (2017) 132–150.

[4] H. T. C. Nguyen, S. Lee, J. Kim, J. Ko, M. Comuzzi, Autoencoders for improving quality of process event logs, Expert Systems with Applications 131 (2019) 132–147.

[5] J. Ko, M. Comuzzi, Detecting anomalies in business process event logs using statistical leverage, Information Sciences 549 (2021) 53–67.

[6] J. Ko, J. Lee, M. Comuzzi, Air-bagel: An interactive root cause-based anomaly generator for event logs., in: ICPM Doctoral Consortium/Tools, 2020, pp. 35–38.

[7] C. E. Ribeiro, L. E. Zárate, Data preparation for longitudinal data mining: a case study on human ageing, Journal of Information and Data Management 7 (2016) 116–116.
[8] M. F. Sani, S. J. van Zelst, W. M. van der Aalst, Repairing outlier behaviour in event logs using contextual behaviour, Enterprise Modelling and Information Systems Architectures (EMISAJ) 14 (2019) 5–1.
[9] K. Böhmer, S. Rinderle-Ma, Multi-perspective anomaly detection in business process execution events, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2016, pp. 80–98.
[10] F. Bezerra, J. Wainer, Algorithms for anomaly detection of traces in logs of process aware information systems, Information Systems 38 (2013) 33–44.
[11] T. Nolle, A. Seeliger, N. Thoma, M. Mühlhäuser, Deepalign: Alignment-based process anomaly correction using recurrent neural networks, in: International Conference on Advanced Information Systems Engineering, Springer, 2020, pp. 319–333.
[12] J. Wang, S. Song, X. Zhu, X. Lin, Efficient recovery of missing events, Proceedings of the VLDB Endowment 6 (2013) 841–852.
[13] W. Song, X. Xia, H.-A. Jacobsen, P. Zhang, H. Hu, Heuristic recovery of missing events in process logs, in: 2015 IEEE International Conference on Web Services, IEEE, 2015, pp. 105–112.
[14] A. Rogge-Solti, R. S. Mans, W. M. van der Aalst, M. Weske, Improving documentation by repairing event logs, in: IFIP Working Conference on The Practice of Enterprise Modeling, Springer, 2013, pp. 129–144.
[15] P. M. Dixit, S. Suriadi, R. Andrews, M. T. Wynn, A. H. ter Hofstede, J. C. Buijs, W. M. van der Aalst, Detection and interactive repair of event ordering imperfection in process logs, in: International Conference on Advanced Information Systems Engineering, Springer, 2018, pp. 274–290.
[16] J. Xu, J. Liu, A profile clustering based event logs repairing approach for process mining, IEEE Access 7 (2019) 17872–17881.
[17] V. Levenshtein, Levenshtein distance, 1965.
[18] J. Carmona, B. van Dongen, A. Solti, M. Weidlich, Conformance checking, Springer, 2018.
[19] D. Schuster, S. van Zelst, W. M. van der Aalst, Alignment approximation for process trees, arXiv preprint arXiv:2009.14094 (2020).
[20] S. J. Leemans, D. Fahland, W. M. Van Der Aalst, Process and deviation exploration with inductive visual miner., BPM (Demos) 1295 (2014).
[21] A. Berti, S. J. van Zelst, W. M. van der Aalst, Pm4py web services: Easy development, integration and deployment of process mining features in any application stack., in: BPM (PhD/Demos), 2019, pp. 174–178.
[22] F. Folino, G. Greco, A. Guzzo, L. Pontieri, Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction, Data & Knowledge Engineering 70 (2011) 1005–1029.