Information Systems 122 (2024) 102349

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Model repair supported by frequent anomalous local instance graphs

Laura Genga [a,*], Fabio Rossi [b], Claudia Diamantini [b], Emanuele Storti [b], Domenico Potena [b]

[a] *Eindhoven University of Technology, Eindhoven, The Netherlands*
[b] *Università Politecnica delle Marche, Ancona, Italy*

ARTICLE INFO

Keywords:
Process mining
Model repair
Subgraph mining

ABSTRACT

Model repair techniques aim at automatically updating a process model to incorporate behaviors that are observed in reality but are not compliant with the original model. Most state-of-the-art techniques focus on the fitness of the repaired models, with the goal of including single anomalous behaviors observed in a log in the form of the events. This often hampers the precision of the obtained models, which end up allowing much more behaviors than intended. In the quest of techniques avoiding this over-generalization pitfall, some notion of higher-level anomalous structure is taken into account. The type of structure considered is however typically limited to sequences of low-level events. In this work, we introduce a novel repair approach targeting more general high-level anomalous structures. To do this, we exploit instance graph representations of anomalous behaviors, that can be derived from the event log and the original process model. Our experiments show that considering high-level anomalies allows to generate repaired models that incorporate the behaviors of interest while maintaining precision and simplicity closer to the original model.

1. Introduction

Today's organizations increasingly rely on the use of information systems to support the execution of their business processes. Some well-known examples are Workflow Management Systems (WMS), Enterprise Resource Planning (ERP), and Business Process Management Systems (BPMS). These systems typically record all past process executions (also termed *cases*) in an *event log*.

Process mining aims at extracting from event logs valuable knowledge about the corresponding processes [1]. There are three main types of process mining, namely: *process discovery*, which aims at automatically distilling a process model from an event log; *conformance checking*, whose goal is to compare a process model against an event log to pinpoint differences between the expected and the actual process behaviors; and, finally, *process enhancement*, aimed to improve and/or to enrich a given process model using information stored in the event log. The present work belongs to the third type of process mining; in particular, we focus on *model repair*, which is a family of techniques aimed at aligning an existing process model M with actual behaviors. More precisely, it aims at automatically building a new model M' able to represent (part of) *anomalous* behaviors, i.e., process executions stored in an event log L but not allowed according to the original model M. Model repair techniques are useful in a number of scenarios. For example, one might need to update a process model that does not properly reflect the reality anymore; employees might find out efficient workarounds that speed up the process; there might exist exceptional situations not properly modeled; and so on.

An important challenge to address when repairing a process model is to generate models of a good *quality*. While different metrics have been defined to assess quality, the fundamental measure is *fitness*, that is a measure of the extent to which the model represents the stored executions, since it would not make much sense to evaluate other quality measures on an underfitting process. However, as observed by [2], focusing solely on fitness can easily lead to models that over-generalize the original ones, including much more behaviors than intended. A promising strategy to mitigate over-generalization is considering structured anomalous behaviors.

The following subsection exemplifies the idea discussing a motivating example.

1.1. Motivating example

Let us consider a simplified version of a loan management process derived from the BPI2012 challenge [3,4]. Fig. 1 shows the process in Petri net notation. The parts in black represent the original model. The process starts with the submission of an application (AS), followed by a first assessment (FA_s, FA_e) to verify the requirements on the applicants and by a fraud check (FC_s, FC_e). For these activities the start and end are explicitly modeled, represented by the "s" and "e" subscripts.

* Corresponding author.
E-mail addresses: l.genga@tue.nl (L. Genga), S1086654@studenti.univpm.it (F. Rossi), c.diamantini@univpm.it (C. Diamantini), e.storti@univpm.it (E. Storti), d.potena@univpm.it (D. Potena).

https://doi.org/10.1016/j.is.2024.102349
Received 22 March 2023; Received in revised form 22 January 2024; Accepted 23 January 2024
Available online 29 January 2024

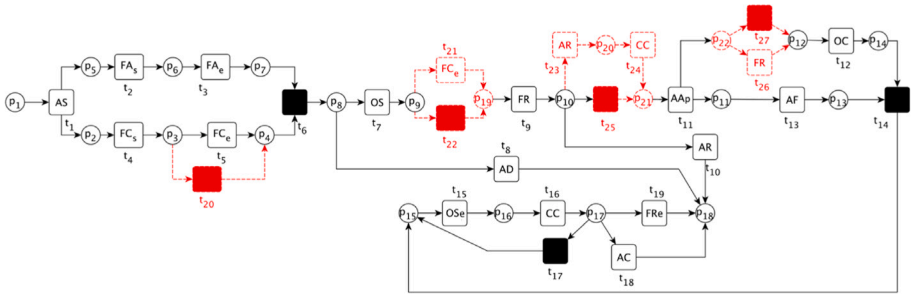


Fig. 1. Loan management process model, original (solid black) and updated to include the discussed anomalous behavior (dashed red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

If the application is not eligible, it is denied (AD); otherwise, a possible offer for the customer is selected (OS). The application is then reviewed by the manager (FR), who can decide to reject (AR) or to approve (AAp) it. If the offer is approved, then the application details are finalized (AF) and an offer is created and customized for the client (OC). The offer is then sent (OSe) and the client is then contacted (CC); it is possible that some negotiations happen, as represented by the loop. If the client does not accept the offer, the procedure is canceled (AC); otherwise, the agreed application is finalized and registered into the system (FRe).

Let us assume that there exist two commonly accepted practices, which however are not represented in the model, and are therefore considered anomalous behaviors. The first one corresponds to a delay of the fraud checking activity, to allow employees to execute other tasks while the checking is not finished yet. However, this is accepted only if the fraud checking step is anyway completed *before* the manager's approval. The second anomalous behavior, instead, represents situations in which an application initially rejected is resumed after contacting the customer. This might happen, for instance, for customers who made some minor mistake in the application and for which the employee wants to proceed without re-starting the process. In this case, the application has to be reviewed again by the manager before finalizing it and creating the offer.

These are examples of structured anomalous behaviors, because they involve a change in the flow relation between two or more activities. In our example, the delay of the fraud checking activity implies a different relation between activity FC_e and activities OS and FR. Similarly, for the resumed application, several flow relations are changed, for instance the execution of AR no longer implies the termination of the process. We call them *high-level* anomalous behaviors, in contrast with *low-level* anomalous behaviors which simply involve a single activity, without taking relations among activities into account.

The daily adoption of these anomalous practices is registered in the log.

Let $\sigma_1 = \langle AS, FA_s, FA_e, FC_s, OS, FC_e, FR, AAp, OC, AF, OSe, CC, FRe\rangle$ and $\sigma_2 = \langle AS, FA_s, FA_e, FC_s, FC_e, OS, FR, AR, CC, AAp, FR, OC, AF, OSe, CC, OSe, CC, FRe\rangle$ be two executions in which respectively the first and the second anomalous behavior occurred.

We hasten to note that structured behaviors have, as a consequence, an impact on the co-occurrence of anomalies registered in the log: the delay of the fraud check activity always results both in the lack of the prescribed FC_e before OS in the log (typically called deleted activity) and in the unexpected presence of FC_e before FR (inserted activity).

Looking at single, low-level anomalies in isolation (i.e., deleted activity *and* inserted activity), state-of-the-art techniques [5] will return the model in Fig. 1, with the additional components highlighted in dashed red.[1] The red components before and after t_9 show the repair performed for the first and second anomalous behavior, respectively. While this repaired model includes the desired behaviors, this solution is not ideal. FC_e can be skipped at every execution, and it is always possible to execute it before the final review by the manager. Similarly, it is always possible to skip/execute the activities AR, CC, with/without executing FR after AAp. These changes introduce more behaviors than needed, which can pave the way to undesirable situations. The reason is that state-of-the-art techniques aim at repairing anomalies independently on each other, without accounting for possible co-occurrence among them. As shown by these simple examples, this can significantly hamper the precision of the updated models, motivating the need for repair techniques tailored to high-level anomalous behaviors, which can include different low-level ones. A first step in this direction has been carried out by [2]. However, this solution is based on the definition of a-priori *change patterns*, which define a set of templates for the anomalies to identify. The behaviors originally represented by the model are then changed according to a tailored set of repairing rules for each pattern. High-level anomalous behaviors not fitting any predefined patterns cannot be detected, with the result that their low-level components are likely to be assigned to different low-level templates and repaired independently from each other. On the other hand, inductive techniques like [5] are able to take into account higher-level behaviors, but they are limited to the discovery of sequences. An example of repair of the loan management process based on the discovering of high-level anomalous behaviors is shown in Fig. 10(b), as a preview of our contribution.

1.2. Paper contribution and organization

Given the issues discussed on the motivating example, the present paper aims at answering the following research questions:

RQ1: How to define a *general* repair methodology to automatically extract structured anomalous behaviors from an event log and generate a repaired model including the extracted behaviors, so to improve fitness while preserving precision and simplicity of the original model?

RQ2: Which are the advantages and limits of such a structured approach with respect to taking into account each low-level anomaly?

To answer RQ1, we propose to model structured anomalies as instance graphs. The discovery can then be performed by exploiting frequent graph mining techniques previously proposed by the authors [4,6]. The issue is then how to process these graphs, in order to incorporate them in the model correctly and effectively. We here propose a novel methodology that comprises the following contributions: (1) a procedure to determine the relevant candidate traces supporting repairing, and the core repairing step that (2) aligns the structured anomalous behavior to the model, (3) individuates the best location where the structured anomalous behaviors should be placed in the model, and (4) converts the graph in the given process model notation and properly merges it

[1] The model of the example has been repaired using the *ModelRepair* plugin implementation available at https://github.com/promworkbench/ModelRepair.

with the original model. In the paper we will consider process models in Petri net notation. It is also worth noting that the new behavior is merged as an alternative branch, thus preserving the behaviors of the original model.

Global repair of a model, which includes all the anomalies in the log, may not reflect the requirements of real-world scenarios. Anomalous behaviors can violate laws and regulations, or some of them may not be in line with strategic business goal. For instance, if an anomaly related to a special treatment for few platinum clients leads to a substantial improvement of return, it can be justifiable to promote it to standard practice, while at the same time refusing another one which applies to every client. The discovery of high-level anomalies makes it simple to define a local repair approach, allowing the user to filter some of them out.

In order to answer RQ2, we perform a comprehensive set of experiments based on synthetic and real-world data, comparing our results with those obtained by state-of-the-art technique [5]. The experiments show that our structured approach leads to repaired models with higher precision and simplicity, while the fitness values remain comparable.

The rest of this paper is organized as follows. Section 2 provides a brief overview on the state-of-the-art; in Section 3 we introduce some definitions used throughout the paper; Section 4 provides the details of the proposed model repair technique; Section 5 describes experiments carried out to validate the approach and test its performance; finally, Section 6 draws some conclusions and delineates future work.

2. Related work

Traditionally, process repairing has mostly been achieved as a manual activity in organizations, often with the support of methodologies for performance measurement and monitoring. In the following, we will briefly discuss the automated and semi-automatic approaches that are more relevant to this work.

The major goal of process repairing is to align the existing, possibly normative model to the reality of daily work practices, thus many work focus on the enhancement of the process in terms of its capability to fit an existing process log. The technique proposed by Fahland et al. [5] is the first in this direction. According to the approach, all anomalies (from non-outlier traces) are considered to be incorporated in the model, although some of them may cause poor performance or can violate laws and regulations. In [5], authors propose a so called naive approach, which repairs a model considering each single anomaly so to maximize fitness, but scoring poorly in terms of precision, i.e. the capability to allow only behaviors that actually occurs in the event log. An advanced version that also caters for precision is then proposed, which takes into account subprocesses, represented by *(maximal) sequences* of log moves. Some post-processing options are also introduced to improve other quality metrics, in particular the simplicity of the model. The present paper shares a similar attention to precision and simplicity, and extends the idea of sequence towards the discovery of more complex structures in the log.

Similarly to the above mentioned work, in [7] a technique aimed to repair the model while improving the fitness as much as possible is proposed. The main differences are that each repair action involves a cost, a repair procedure must not exceed a predefined total cost, and an optimal strategy is determined to minimize the costs of the repairs and maximize the fitness of the repaired model.

Other approaches are more tailored towards precision. In particular, the work by Dees et al. [8], introduces an approach to assess the impact of anomalies on process KPIs, to repair only those with a positive impact and which do not violate a-priori defined constraints. This idea of a local repair is common to the approach presented in this paper. The technique, however, applies the repair approach of [5] and is thus designed for punctual, low-level alignment moves, and does not take into account high-level anomalous patterns. Furthermore, the way low-level anomalies are selected makes it not comparable to our approach.

High level “change patterns”, representing typical anomalous behaviors, are introduced in [2] to drive the repair procedure in an incremental way, providing suggestions about how to implement the identified pattern(s) in the current model. However, this solution can only detect behaviors described by a-priori designed patterns, for which tailored repairing rules have been developed. Furthermore, some of the repair rules replace the original behavior, rather than adding the anomalous one as an alternative. We would like to point out that the prototype available for this technique presents some scalability issues which made a thorough comparison with our approach not feasible.

In the same vein, the work of Adriansyah et al. [9] introduces a technique to guide the users in manually adding high-level anomalous patterns involving a set of activities of interest to a process model, in particular swaps and replacements, and introduces an extension of alignment-based conformance checking approach to detect these high-level anomalies. The work deals with a very special use case related to the bypass of security mechanisms in IT systems in case of emergency.

In [10], a pre-processing step is performed to identify the relevant traces for the repair. By relying on Inductive Logic Programming and analyzing concept drifts, the model is decomposed by experts in an unchangeable part, which is assumed to be correct and therefore is not allowed to change, and a changeable part. Then, traces in the log are identified as positive (i.e., behavior that is acceptable) and negative (behavior that is forbidden). Only the positive part is used to repair the changeable part of the model distinguishing between adding new activities and removing infrequent ones and taking into account a minimality criterion. The approach takes into account, as we do, the fact that a global repair of the model with all the detected anomalies may not be desirable or acceptable, however in this case domain experts are in charge of model decomposition, while in our case we assume the expert can accept or refuse the discovered anomalies.

A more recent set of approaches based on Logic Petri Nets (LPNs) demonstrates the capability to improve the fitness and the precision of the repaired model with respect to traditional techniques by referring to more expressive nets with logic constraints. Different techniques have been experimented, e.g., based on alignment [11], on token-replay [12] or others (see [10] for a discussion).

At the best of our knowledge no process repair technique combines both ideas of local repair and discovery of high-level anomalies to balance fitness and precision.

Other works face different approaches and of different quality measure. For instance, the work by by Buijs et al. [13] proposes process discovery techniques by genetic algorithms which mediate different model quality criteria besides fitness and precision. In particular, since the method discovers a new model instead of updating the original one, model similarity is of tantamount importance. Mitsyuk et al. [14] adopt a similar idea, but they decompose the event log and the process model in order to identify process fragments affected by the anomalies and mine a new model only for that process portion(s). By considering a collection of models instead of an event log as input, in [15] authors propose to repair a process model to make it similar to a collection of process models, trying to minimize the edit distance to all models in the collection.

Model repair can also be interpreted in different settings. For instance, it can also be seen as simplifying the model while allowing for the same behavior, as proposed by Fahland et al. [16]. Similarly, the precision of an existing model w.r.t. the event log can be enhanced by automatically introducing non-local constraints [17]. Other approaches focus on ensuring run-time flexibility, allowing to adapt the model at run-time and thus creating individual models for different process executions, at the same time ensuring consistency and avoiding run-time errors [18]. Yet another interpretation of process repair consists in automatically correct modeling errors, e.g., to remove deadlocks [19, 20].

Some alternative approaches have also been investigated, e.g. directly focusing on event logs instead of enhancing an existing process model. Authors in [21] propose a methodology aimed to show improvements at the instance level. This is done by changing the event log, for instance by reordering activities or changing the resource allocations to improve the process in terms of some indicators like flow time or costs, and the derived log is then checked to verify whether it is still compatible with the original one or not. Similarly, in [22], a semi-automated approach aims to improve the business performance of processes by deriving decision criteria from the experience gained through past process executions.

3. Preliminaries

In this section, we introduce some core definitions used throughout the paper.

Definition 1 (*Labeled Petri Net*). A *labeled Petri net* is a tuple (P, T, F, A, ℓ) where P is a set of *places*, T is a set of *transitions*, $F \subseteq (P\times T)\cup(T\times P)$ is the *flow relation* connecting places and transitions, A is a set of *labels* for transitions, and $\ell : T \rightarrow A$ is a *function that associates a label with every transition in* T. The notation ${}^\bullet x$ ($x^\bullet$), $x \in P\cup T$ is adopted to denote input (output) relations among places and transitions: ${}^\bullet x = \{y|(y,x) \in F\}$ ($x^\bullet = \{y|(x,y) \in F\}$).

Labels associated with transitions represent process activities, namely well-defined tasks that have to be performed within the process. It is useful to introduce a subset $T_H \subseteq T$ of *hidden transitions*. Hidden transitions are used for routing purposes and are not associated with process activities. In this case, the definition of the labeling function is modified as follows $\ell : T \setminus T_H \rightarrow A$. Places represent pre-conditions (post-conditions) for the execution of activities. Places can contain zero or more tokens, representing the enabling of a condition. A distribution of tokens over places is called a *state*. In classical nets the state can be expressed as a function $m : P \rightarrow \mathbb{N}$ that assigns to each place the number of tokens in that place. This function is called a *marking* of the net.

The execution semantics of a Petri Net is defined by a *firing rule*, that describes the conditions for firing a transition, and the consequent change of state of the Petri Net. According to the rule, a transition t is enabled in a given marking m if all its pre-conditions are enabled: $\forall x \in {}^\bullet t, m(x) > 0$. The firing of an enabled transition produces a new state defined by the marking m' such that $\forall x \in {}^\bullet t, y \in t^\bullet, m'(x) = m(x) - 1, m'(y) = m(y) + 1, m'(p) = m(p)\ \forall p \notin {}^\bullet t \cup t^\bullet$. We denote it as $m \xrightarrow{t} m'$. The subsequent firing of transitions defines the evolution of the system, typically from an initial marking m_0.

Definition 2 (*Marking Reachability*). Let $M = (P, T, F, A, \ell)$ be a Petri Net. A marking m' is *one step reachable* from a marking m, if there exists a transition t such that $m \xrightarrow{t} m'$. A marking m' is said to be reachable from m if there is sequence of transitions $\rho = \langle t_1, t_2, \dots, t_n\rangle$ such that $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \dots \xrightarrow{t_n} m'$. ρ is also called a *firing sequence* and we will denote by $m \xrightarrow{\rho} m'$ the reachability of m' from m through the firing sequence ρ.

The set of reachable markings of M, denoted by $\mathcal{R}(M)$, is the set of all markings reachable from the initial marking m_0: $\mathcal{R}(M) = \{m|\exists\rho = \langle t_1, t_2, \dots, t_n\rangle \in T^* : m_0 \xrightarrow{\rho} m\}$. From the notion of reachable marking, the reachability graph of a Petri Net can be defined.

Definition 3 (*Reachability Graph*). Let $M = (P, T, F, A, \ell)$ be a Petri Net. The *reachability graph* of M, denoted $RG(M)$ is a graph $RG(M) = (Q, \Delta, R, m_0)$ where $Q = \mathcal{R}(M)$ is the set of nodes representing the set of all markings reachable from the initial marking m_0, $\Delta = A$, $R \subseteq Q\times\Delta\times Q$ is the set of arcs connecting two markings labeled with the activity associated to the transition: $R = \{(m, \ell(t), m')|\exists t \in T : m \xrightarrow{t} m'\}$.

An interesting sub-class of labeled Petri Nets are Workflow Nets (WF-nets).

Definition 4 (*Workflow Net*). A labeled Petri Net is a WF-net if and only if:

- there is a distinguished initial place $s \in P :\ {}^\bullet s = \emptyset$
- there is a distinguished final place $f \in P :\ f^\bullet = \emptyset$
- every other place and transition belongs to a path from s to f.

WF-nets have been introduced to conveniently model the workflow of business processes. The single initial and final places correspond to real-life situations in which processes have specific starting points and specific end points, while the third condition models the fact that any task can be executed and actually contributes to the completion of at least some process executions.

Well formed business processes are typically associated with the notion of *soundness*. We refer to [23] for details. In this setting, the starting point of the process is represented by the initial marking $m_0 : m_0(s) = 1, m_0(p) = 0\ \forall p \in P, p \neq s$, and the end point by the final marking $m_f(f) = 1, m_f(p) = 0\ \forall p \in P, p \neq f$. Note that, in WF-nets, markings reduce to indicator functions. The rationale under this is that the initial marking represents the initial state of a specific execution.

Specific executions of a process are called *process instances* or *cases*, and are typically recorded in logs in the form of traces. More precisely, the execution of an activity generates an *event*, which is captured by a logging system and recorded in the event log at the time of its occurrence.

Definition 5 (*Event, Trace, Log*). Let $\mathcal{A}$ be the set of all possible activity names. An *event* e represents a unique recorded execution of an activity $a \in \mathcal{A}$. For an event e, $act(e) \in \mathcal{A}$ denotes the activity associated to e. The set of all possible events is denoted by $\mathcal{E}$. A trace $\sigma = \langle e_1, \dots, e_n\rangle$ is a sequence of events $e_i \in \mathcal{E}, i = 1, \dots, n$ where $\forall i < j$, e_i has been executed before e_j. A *log* $\mathcal{L}$ is a multi-set of traces.

A well-known issue of sequential traces is that they hide possible concurrency among activities. To address this issue, in the present paper we propose to take into account process instances directly, representing them as graphs. *Instance graphs* [6] are directed acyclic graphs where nodes represent events, and arcs connect events only if there exists a strict execution constraint between the associated activities, as formalized by the notion of *causal relation.*

Definition 6 (*Causal Relation*). A *Causal Relation* (CR) is a relation on the set of activities, $CR \subseteq \mathcal{A}\times\mathcal{A}$. Given activities $a_1, a_2 \in \mathcal{A}$, $a_1 \rightarrow_{CR} a_2$ denotes $(a_1, a_2) \in CR$.

Elements of CR represent the order of execution of a pair of activities of a process. Indeed, $a_1 \rightarrow_{CR} a_2$ states that a_2 cannot be executed until a_1 is terminated; in other words, the execution of a_2 *depends* on the execution of a_1.

Causal relations can be determined from existing process models. To this end, given a WF-net (P, T, F, A, ℓ) representing a process model, we introduce the notion of *direct path* between transitions t_1, $t_2 \in T$ as follows: a direct path between t_1 and t_2, denoted by $dp(t_1, t_2)$, exists if and only if $\exists p \in P$ s.t. $(t_1, p) \in F \wedge (p, t_2) \in F$. It should be noted that t_1, t_2 can be either labeled or hidden transitions. In this setting, $a_1 \rightarrow_{CR} a_2$ if $\exists t_1, t_2 \in T \wedge l(t_1) = a_1 \wedge l(t_2) = a_2 \wedge (\exists dp(t_1, t_2) \vee (\exists\{t_{h_1}, \dots, t_{h_n}\} \subseteq T_H \wedge \exists dp(t_1, t_{h_1}) \wedge \exists dp(t_{h_n}, t_2) \wedge \forall i \in \{1, \dots, n-1\}, \exists\ dp(t_{h_i}, t_{h_{i+1}})))$.

Informally, this definition considers as a causal relation a direct path between two labeled transitions, in which at most a sequence of hidden transitions is allowed.

Definition 7 (*Instance Graph*). Let $\sigma = \langle e_1, \dots, e_n\rangle \in \mathcal{L}$ be a trace and CR be a causal relation on the set of activities $\{act(e_1), \dots, act(e_n)\}$. An *Instance Graph* (or IG) γ_σ of σ is a directed acyclic graph (E, W, l) where:

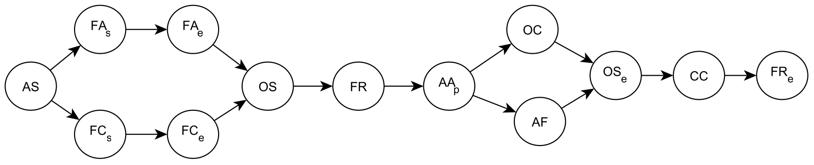


Fig. 2. The instance graph of σ.

- $E = \{e \in \sigma\}$ is the set of events in σ. Each event corresponds to a node in the graph;
- $W = \{(e_h, e_k) \in E \times E \mid h < k \wedge act(e_h) \rightarrow_{CR} act(e_k)\}$ is the set of edges, defining a strict partial order[2];
- $l : E \rightarrow \mathcal{A}$ with $l(e) = act(e)$ is a function labeling each node with the corresponding activity.

Example 1. Let us consider the set $CR = \{(AS, FA_s)$, (FA_s, FA_e), $(AS,\ FC_s)$, (FC_s, FC_e), (FA_e, OS), (FC_e, OS), (FA_e, AD), (FC_e, AD), (OS, FR), $(FR,\ AR)$, (FR, AAp), (AAp, AF), (AAp, OC), (OC, OSe), (AF, OSe), $(OSe,\ CC)$, (CC, OSe), (CC, AC), $(CC, FRe)\}$ derived from the Petri net in Fig. 1 and the trace $\sigma = \langle AS, FA_s, FA_e, FC_s, FC_e, OS, FR,\ AAp, OC, AF, OSe, CC,\ FRe\rangle$. For the sake of simplicity, hereafter we use event activities to represent a trace. Fig. 2 shows the IG corresponding to the trace.

Note that IGs contain only sequential and concurrent relations among events, and they do not include loops and choices. Indeed, choices do not exist in a trace, since in each single execution choices have already been made. Executing a loop in the model gives rise to different events in the trace with the same labels, leading to an IG with different nodes corresponding to the same activity.

We can define the set of "legal" process executions according to a process model as the set of IGs associated with the traces generated by all possible firing sequences of the WF-Net representing it.

Definition 8 (*IG Set of a WF-Net*)**.** Let $M = (P, T, F, A, \ell)$ be a WF-Net with initial marking m_0 and final marking m_f, let $CR \subseteq A \times A$ be the causal relation defined over M, let $\rho = \langle t_1, \ldots, t_n\rangle \in T^* : m_0 \xrightarrow{\rho} m_f$ be a firing sequence, let $\rho_{_A} = \rho|_{T \setminus T_H}$[3] be the projection of ρ over the set of labeled transitions . Let also define $\sigma_M = \langle e_1, \ldots, e_n\rangle$ as the trace where each event e_i corresponds to the firing of the ith transition in $\rho_{_A}$, with $act(e_i) = \ell(t_i)$. The IG set of M, denoted by $\mathcal{IG}(M)$, is the set of Instance Graphs of all possible σ_M over the causal relation CR.

Given a set of instance graphs, it is possible to derive a set of *local IG* (LIG) representing portions of process behaviors.

Definition 9 (*Local Instance Graph*)**.** Let $\gamma = (E, W, l)$ be an IG. $\gamma' = (E', W', l')$ is a Local Instance Graph (LIG) of γ if (a) $E' \subseteq E$, (b) $W' \subseteq W$ (c) l' is l restricted to domain E', and (d) γ' is a connected graph.

Subgraph mining techniques are typically used to extract common subgraphs from a set of graphs by incrementally generating a set of candidate subgraphs and then looking for the *embedding* of these candidates in the graphs set.

Definition 10 (*Subgraph Isomorphism, Embedding*)**.** A labeled graph $\gamma = (E,\ W,\ l)$ is *isomorphic* to another graph $\zeta = (E_\zeta, W_\zeta, l_\zeta)$ if and only if a bijection $f : E \rightarrow E_\zeta$ exists such that (i) $\forall e_i \in E\ \ l(e_i) = l_\zeta(f(e_i))$, and (ii) $\forall (e_i, e_j) \in W \iff (f(e_i), f(e_j)) \in W_\zeta$. The graph $\psi = (E_\psi, W_\psi, l_\psi)$ is *subgraph isomorphic* to γ if there exists γ' such that γ' is a subgraph of γ, and ψ is isomorphic to γ'. γ' is said to be an *embedding* of ψ in γ.

The approach proposed in this paper aims at repairing a given process model by introducing anomalous behaviors represented in terms of LIGs. To place these behaviors in the model, we first have to detect the position in the process in which they occur. To this end, given a LIG γ' and a trace σ we need to detect one or more *trace embedding(s)* of γ' in σ. Informally, this means to extract the subsequence(s) of σ complying with the ordering relations shown in γ'. A formal definition of trace embedding is introduced below.

Definition 11 (*Trace Embedding, Extended Embedding*)**.** Let $\sigma \in \mathcal{L}$ be a trace and γ_σ be the IG of σ. Let γ be a graph of which an embedding γ' exists in γ_σ. The *trace embedding* of γ in σ w.r.t. γ' is the sequence $s = \sigma|_{E'}$. The *extended embedding* σ_e of γ in σ w.r.t. γ' is the contiguous subsequence of σ whose first and last elements are equal to the first and last elements of s, respectively.

Example 2. Let us consider the trace σ and the CR introduced in Example 1. Fig. 3(a) shows a possible LIG extracted from the IG in Fig. 2. The trace embedding of this LIG in σ is $s = \langle AS, FA_s, FC_s, FC_e, OS\rangle$, while the extended embedding is $\sigma_e = \langle AS, FA_s, \underline{FA_e}, FC_s,\ FC_e, OS\rangle$.[4] Note that the event FA_e does not belong to the trace embedding; however, it belongs to σ_e, since it occurs in between the events included in the embedding s.

Hereafter, we will denote by LIG_{σ_e} the graph corresponding to the extended embedding σ_e of a LIG in σ (see Fig. 3(b)). It is straightforward to show that LIG is a subgraph of LIG_{σ_e}, and σ_e is the trace embedding of LIG_{σ_e} in σ.

We refer to the concept of alignment to assess the conformance of a trace to a process model. An *alignment*, here indicated with the symbol η, shows a possible correspondence between a trace of an event log and a firing sequence of the WF-net. For example, a possible alignment between the original Petri net in Fig. 1 and the trace $\sigma_1 = \langle AS, FA_s, FA_e, FC_s, OS, FC_e, FR, AAp, OC, AF, OSe,\ CC, FRe\rangle$ is shown in Fig. 4. The first two rows refer to the model . In particular, the first row shows activities corresponding to transitions of the firing sequence, which are shown in the second row. The third row shows the sequence of activities associated to events in the trace. We use the symbol τ as label for hidden transitions. When the trace perfectly fits with the process model, each activity in the first row occurs also in the third row, in the same position; otherwise, a "no-move" symbol $\gg$ is inserted. Hereafter, we define a *move* the pair $\eta[i] = (t_i, a_i)$, where t_i is the transition in the second row and a_i is the activity in the third row occurring in the ith position of η. There can be three possible kinds of moves: (a) a *synchronous move* if a_i is an event label in the trace and t_i is a transition in the model with the same label, (b) a "move-on-log", if t_i is $\gg$ and (c) a "move-on-model", if a_i is $\gg$.

Note that a move-on-log represents an inserted activity. Indeed, it means that an activity has occurred in a position not allowed by the model. A move-on-model represents a deleted activity; it means that a certain activity should have occurred according to the model and did

[2] Given a set Φ, a strict partial order over Φ is a binary relation $\prec \subseteq \Phi \times \Phi$ that is *(i)* irreflexive, i.e. $\phi \nprec \phi$, *(ii)* asymmetric, i.e. if $\phi \prec \phi'$, then $\phi' \nprec \phi$, and *(iii)* transitive, i.e. if $\phi \prec \phi'$ and $\phi' \prec \phi''$, then $\phi \prec \phi''$, for any $\phi, \phi', \phi'' \in \Phi$.

[3] The symbol $|_X$ stands for the *projection* over the set X(i.e., only elements belonging to X are kept).

[4] Underlined activities represent those added to the trace embedding.

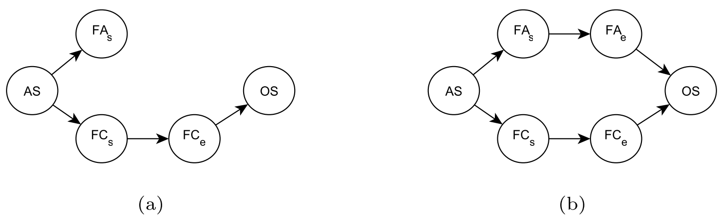


Fig. 3. (a) a possible LIG extracted from the IG in Fig. 2, (b) LIG_{σ_e}.

$$\eta_1 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\mathsf{AS} & \mathsf{FA_s} & \mathsf{FA_e} & \mathsf{FC_s} & \mathsf{FC_e} & \tau & \mathsf{OS} & \gg & \mathsf{FR} & \mathsf{AAp} & \mathsf{OC} & \mathsf{AF} & \tau & \mathsf{OSe} & \mathsf{CC} & \mathsf{FRe} \\
(\mathsf{t_1}) & (\mathsf{t_2}) & (\mathsf{t_3}) & (\mathsf{t_4}) & (\mathsf{t_5}) & (\mathsf{t_6}) & (t_7) & & (\mathsf{t_9}) & (\mathsf{t_{11}}) & (\mathsf{t_{12}}) & (\mathsf{t_{13}}) & (\mathsf{t_{14}}) & (\mathsf{t_{15}}) & (\mathsf{t_{16}}) & (\mathsf{t_{19}}) \\
\hline
\mathsf{AS} & \mathsf{FA_s} & \mathsf{FA_e} & \mathsf{FC_s} & \gg & \gg & \mathsf{OS} & \mathsf{FC_e} & \mathsf{FR} & \mathsf{AAp} & \mathsf{OC} & \mathsf{AF} & \gg & \mathsf{OSe} & \mathsf{CC} & \mathsf{FRe}
\end{array}$$

Fig. 4. Alignments of σ_1 and the Petri net of the original loan management process shown in Fig. 1.

not occur in reality. Such cases represent *low-level anomalous behaviors* of the trace with respect to the process model.

Finally, we can provide the definition of structured, or high-level anomalous behavior as follows:

Definition 12 (*High-level Anomalous Behavior*). Let $\mathcal{IG}(M)$ be the IG set of the WF-Net M. Let γ_σ be the IG of a trace σ. A high-level anomalous behavior is defined as a LIG γ' of γ such that an embedding of γ' in the set $\mathcal{IG}(M)$ does not exist.

The definition allows for many high-level anomalous behaviors, also with inclusion relations among them. In the following we will see how smallest most relevant high-level anomalous behaviors can be discovered from a process logs.

4. Methodology

In this section, we introduce our model repair methodology based on anomalous local instance graphs. Given a process model M and an event log $\mathcal{L}$ tracking the process executions, we will describe the steps that return a new process model M' incorporating the behavior of a single LIG. The reason is two-fold. On the one hand, in line with existing local repair approaches, in our setting we assume the possible presence of a user that decides which anomalies should be added. This makes it possible to both discard those violating laws and regulations and, in order to keep the model as simple as possible, choose only those corresponding to some strategic business goal. On the other hand, not having to carry out a complete repair, it makes sense to devise a simple procedure, which can be iterated to integrate other selected anomalies if necessary. In the absence of expert users and other domain-driven criteria, a notion of relevance can be, and will be, adopted as the criterion for LIG selection.

The methodology assumes that every inserted activity must correspond to a process activity. This assumption eliminates the possibility of including noise or errors with the repair, and consequently the necessity of introducing a final verification of the semantic correctness of the repaired model, or a preliminary pre-processing and cleaning of the log.

At high level, the methodology is composed of three steps (see Fig. 5):

- Anomalous LIGs extraction: in this step, a set of anomalous LIGs is extracted from the event log, among which the user selects the one to include. The step exploits an anomaly discovery technique previously proposed by the authors [4], that will be briefly described here for completeness;
- Trace selection: one trace is selected as a guide to repair from the set $\mathcal{L}_{LIG} \subseteq \mathcal{L}$, i.e., the set of traces involving the extended embedding of the LIG. In particular, we select one of the traces with the shortest extended embedding of the LIG, to mitigate the influence of other possible anomalies occurred within the trace but not belonging to the LIG;
- LIG integration: the step is devoted to extending the model with the selected LIG. In detail, this step aligns the structured anomalous behavior to the model; individuates the best location where the structured anomalous behaviors should be placed in the model; and converts the graph in Petri Net notation and properly merges it with the original model.

In the next subsections, the details for each step are given.

4.1. Anomalous LIGs extraction

The anomaly discovery technique [4] takes as input a process model representing the expected process behavior, and an event log storing a set of historic executions of the process. In the following we describe the two main steps carried out, namely *Relevant subgraph mining* and *Anomalous subgraph extraction*.

4.1.1. Relevant subgraph mining

This step is aimed at (a) converting each sequential trace in an Instance Graph (IG), and (b) extracting the most relevant subgraphs.

Instance graph building. First, an IG is built for each trace as in Definition 7. In the presence of non-compliant events, however, this procedure generates anomalous, low-quality IGs. As an example, let us consider σ_1 discussed in Section 1. Fig. 6 shows the IG γ_1 built taking in input the causal relation set CR extracted by the Petri net in Fig. 1. It is straight to see that these IG provide poor quality model for the corresponding process execution. In terms of structure, these anomalies lead to generate either disconnected graphs and/or graphs whose connections among nodes do not reflect the temporal order of occurrence of the events. In terms of semantics, these models over-generalize the process behavior, allowing for much more behaviors than what is observed in the log. For example, the only execution constraint for FC_e, in Fig. 6, is to be executed after FC_s. Indeed, according to this IG, FC_e can be executed in any order with respect to the remaining activities, thus generating traces also very different from σ_1.

To deal with the aforementioned issues, in [6] an *IG repairing* procedure is applied to IGs corresponding to anomalous traces, which transforms them in graphs capable of also representing the anomalous traces without over-generalizing. To this end, at first, anomalous traces (and, hence, IGs) are recognized in the event log by means of a conformance checking technique [24], which returns the positions low-level anomalies and their kind (i.e., move-on-model and move-on-log). Then, tailored rules are applied for repairing IGs with deleted and inserted events. For deleted events, the repair is to identify the nodes that should have been connected to the deleted activity and connect them properly. For each inserted event, taking into account the causal relationships between the predecessors and successors of that event, we

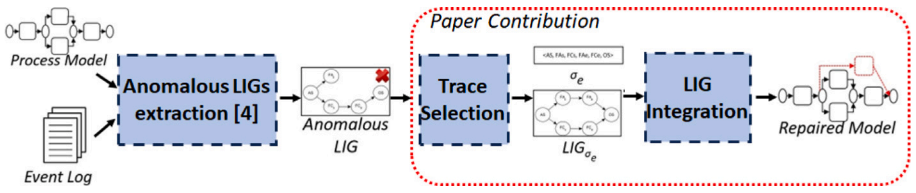


Fig. 5. Overview of the methodology.

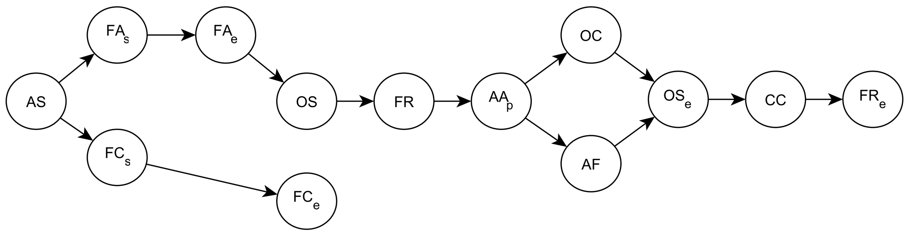


Fig. 6. IG γ_1 built for σ_1 according to Definition 7.

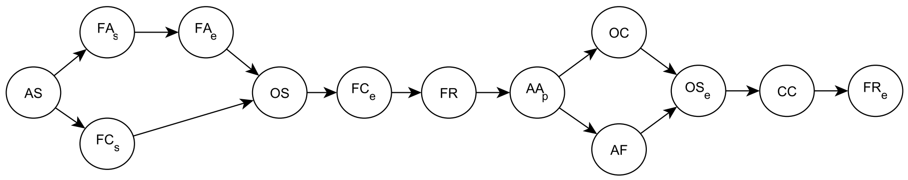


Fig. 7. Repaired IG built for σ_1 (γ_1).

identify the nodes corresponding to the events that occurred before and after the inserted event. Then, we modify the IG in order to connect the entered event to the identified nodes. Fig. 7 shows the outcome of the repairing procedure for the IG corresponding to σ_1. The repairing for γ_1 involves a combination of a deletion repairing (between FC_s and OS) and an insertion repairing (between OS and FR). It is worth noting that the repaired graph does not fulfill anymore Definition 7 with respect to the original casual relation set CR; however, it still fulfills the definition according to the new set of causal relation CR', obtained by extending CR to include all the pair of activities linked by edges added/modified during the repairing procedure (e.g., for γ_1 we add to CR the pairs $(FC_s, OS), (OS, FC_e), (FC_e, FR)$). We refer to [6] for additional details.

Subgraph mining. Once the set of Instance Graphs is built and repaired, a Frequent Subgraph Mining technique (FSM) is applied to extract the most relevant LIGs. In our approach, we relate the relevance of a LIG both to all its occurrences in a graph set and its size. In other words, given two subgraphs with the same occurrence frequency but different sizes, we are interested in the largest. This is motivated by the fact that we expect to derive a larger amount of knowledge from it. The size of a graph can be represented in terms of its Description Length (DL), i.e., the number of bits needed to encode its representation, computed as the sum of the number of bits needed to encode its nodes and the number of bits needed to encode its edges (further details on DL are provided in [25]). For this reason, we exploit the SUBDUE algorithm [26] which is based on DL minimization. We refer to [4] for a detailed description of LIGs extraction by means of SUBDUE. It is worth noting, however, that any other FSM algorithm can be exploited as well.

4.1.2. Anomalous subgraph extraction

The second step of the approach aims to determine among the subgraphs identified in the previous step those that do not fit the given process model. To this end, a tailored algorithm is introduced, the *Subgraph Conformance Checking* (SCC) algorithm, whose core idea is that, given a subgraph γ and the reachability graph $RG(M)$ of a Petri net M, it is possible to determine whether γ is "compliant" or "non-compliant" (i.e., anomalous) with respect to $RG(M)$ by replaying γ over $RG(M)$. Interested readers are referred to [4] for additional details.

It is worth noting that usually *inclusion relations* exist among the subgraphs returned by subgraph mining techniques. Namely, it is likely to have one or more subgraphs involved in larger subgraphs. Subgraphs related by an inclusion relation are highly correlated, thus yielding redundant information.

In this work, we are interested in non-compliant subgraphs that are *representative* of relevant anomalous behaviors, namely all those subgraphs γ_i such that *(i)* γ_i captures a non-compliant behavior, and *(ii)* there does not exists any other subgraph $\gamma_j \subset \gamma_i$ such that the Description Length computed for γ_j is greater than that of γ_i.

Representative subgraphs are the smallest and most relevant non-compliant LIGs representing high-level anomalous behaviors. They allow us to define patterns representing most relevant anomalies, without incurring in the redundancy issue previously discussed. These LIGs are identified by analyzing inclusion relations among non-compliant subgraphs to extract those subgraphs fulfilling requirement (ii).

As anticipated before, we assume the presence of a user that chooses among the set of LIGs which have to be added. In particular, the next steps describe the integration of a single anomaly. The procedure can be iterated to integrate other selected anomalies. In the rest of the paper, we will assume that LIG relevance is the criterion guiding the selection.

4.2. Trace selection

In order to determine the candidate trace for repair, we select the IGs containing the LIG chosen in the previous step, and we compute the alignment of the corresponding traces with respect to the model to be repaired using off-the-shelf techniques [24]. Then, IGs are ordered based on the graph matching cost between the IG of the trace and the LIG, and the first one is selected. This allows to consider the trace with the minimum number of additional anomalies. To this end, we exploit a graph matching algorithm based on the well-known notion of description length, introduced in the previous step [25]. The algorithm

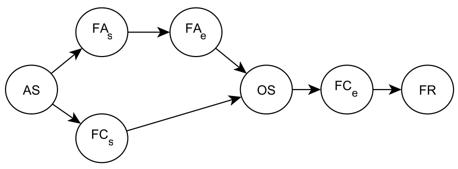


Fig. 8. LIG extracted for the process in Fig. 1.

computes the cost of transforming the larger of the input graphs into the smaller, and returns the cost and the mapping of vertices in the larger graph to vertices in the smaller graph. It should be noted that, while the subgraph matching problem is known to be NP-complete, here node labels are considered to mitigate the computational cost.

Example 3. Let us consider the loan management process of Fig. 1. Let us assume that by applying the LIG detection approach discussed in Section 4.1 we extracted LIG$_1$ shown in Fig. 8. LIG$_1$ shows that the activity FC_e has been delayed, and executed after OS. Let us consider the traces $\sigma_1 = \langle AS,\ FA_s,\ FA_e,\ FC_s,\ OS,\ FC_e, FR, AAp, OC, AF, OSe, CC, FRe\rangle$ and $\sigma_3 = \langle AS,\ FA_s,\ FA_e,\ FC_s,\ AR,\ OS,\ FC_e,\ FR,\ AAp,\ OC, AF, OSe, CC, FRe\rangle$.

Their corresponding extended embeddings for the considered LIG are $\sigma_{e_1} = \langle AS, FA_s, FA_e, FC_s, OS, FC_e, FR, \rangle$, $\sigma_{e_3} = \langle AS,\ FA_s,\ FA_e,\ FC_s, \underline{AR},\ OS,\ FC_e,\ FR\rangle$. Fig. 9 shows the graphs LIG$_{\sigma_{e_1}}$ and LIG$_{\sigma_{e_3}}$ related to the two extended embeddings. It is straight to see that while $LIG_{\sigma_{e_1}}$ is identical to the LIG to add, $LIG_{\sigma_{e_3}}$ differs for the presence of the path from AS to OS through AR. Therefore, while both traces involve an embedding of LIG$_1$, σ_1 is the trace at minimum graph matching cost, and will be the one returned by the trace selection step.

4.3. LIG integration

This subsection describes the core step of the approach. Given the selected trace σ for the repair, we extract the extended embedding σ_e of the LIG to be added to the model in σ and we create the corresponding LIG$_{\sigma_e}$. In case of multiple embeddings, we consider only the first occurrence. This choice is driven by simplicity reasons. As a matter of fact, the integration of the corresponding behavior in more than one point of the model would make the description of the algorithm significantly more complex. Hence, we have decided not to deal with this special case in this paper and to repair only the first occurrence, also in the light of the fact that, in principle, with the present formulation multiple integrations are still possible by iterating the procedure. Then, we compute the alignment η between σ and M. Algorithm 1 describes the methodology to integrate LIG$_{\sigma_e}$ into the process model M.

To simplify the integration procedure, we ensure that the LIG$_{\sigma_e}$ has a unique start event and a unique end event. This type of graph is obtained by adding a node at the beginning (end) of the graph that connects to all start (end) nodes of LIG$_{\sigma_e}$. This is done by the function $addStartEnd$ (see Algorithm 2), which, if necessary, creates and adds a node $start_{new}$ and/or a node end_{new} to the graph, also adding the matching event(s) at the beginning and/or at the end of the extended embedding (line 1). Note that the added nodes are labeled as "hidden", since they do not correspond to actual activities in the process. The function then returns the (modified) graph, the (modified) extended embedding, and the start and the end nodes of the graph. Then, we derive σ_M, that is the projection of the alignment η on the model M; namely, it contains all transitions in M which are in η (line 2). Hence, σ_M conforms to the model.

Lines from 3 to 11 are devoted to find the places of the process model M and events of LIG$_{\sigma_e}$ that will be connected to each other. In particular, the function *findLinks* (see Algorithm 3) scrolls the alignment η until it finds the first non-synchronous move occurring in σ_e. If a

Algorithm 1: LIG integration

```
Input  : The graph LIG$_{\sigma_e} = (E, W, l)$, the alignment $\eta$ of the
         trace $\sigma$ over the process model $M = (P, T, F, A, \ell)$, the
         extended embedding $\sigma_e = \langle e_1^*, \dots, e_n^*\rangle$ of LIG in $\sigma$
Output: Repaired model $M'$
1  $((E, W, l), \sigma_e, start_{new}, end_{new}) = addStartEnd(\text{LIG}_{\sigma_e}, \sigma_e)$;
2  $\sigma_M = \eta\,|_T$;
3  $(start_{LIG}, posStart_{LIG}, start_M, posStart_M) = findLinks(\sigma_e, \eta)$;
4  $\sigma'_e = inv(\sigma_e)$;            /* inv() reverses a sequence */
5  $pos = scan(\eta, \sigma_e)$;
6  $\eta' = inv(left(\eta, pos))$;
7  $(end_{LIG}, posEnd'_{LIG}, end_M, posEnd'_M) = findLinks(\sigma'_e, \eta')$;
8  $posEnd_M = length(\eta'\,|_T) - posEnd'_M + 1$ ;
9  $posEnd_{LIG} = length(\sigma_e) - posEnd'_{LIG} + 1$ ;
10 $P_S = status(left(\sigma_M, posStart_M), M)$;       /* status($\sigma, M$)
    returns places of M with a token after the
    firing of transitions in $\sigma$ */
11 $P_E = status(left(\sigma_M, posEnd_M - 1), M)$;
12 $E' = \{\sigma_e[i] \mid posStart_{LIG} \leqslant i \leqslant posEnd_{LIG}\}$;
13 $W' = \{(e_i, e_j) \in W \mid e_i \in E' \wedge e_j \in E'\}$;
14 $((E, W, l), \sigma_e, start_{LIG}, end_{LIG}) = addStartEnd((E', W', l), \sigma_e|_{E'})$;
15 $T_{LIG} = \ell_{LIG} = \emptyset$;
16 forall $e \in E$ do
17     $map(e) = create\_transition()$;
18     $T_{LIG} = T_{LIG} \cup \{map(e)\}$;
19     if $l(e) \neq$ "$hidden$" then
20         $\ell_{LIG} = \ell_{LIG} \cup \{(map(e), l(e))\}$;
21 $P_{LIG} = F_{LIG} = \emptyset$;
22 forall $(e_i, e_j) \in W$ do
23     $p_{ij} = create\_place()$;
24     $P_{LIG} = P_{LIG} \cup \{p_{ij}\}$;
25     $F_{LIG} = F_{LIG} \cup \{(map(e_i), p_{ij}), (p_{ij}, map(e_j))\}$;
26 $T' = T \cup T_{LIG}$; $P' = P \cup P_{LIG}$; $F' = F \cup F_{LIG}$;
27 forall $p' \in P_S$ do
28     $F' = F' \cup \{(p', map(start_{LIG}))\}$;
29 forall $p'' \in P_E$ do
30     $F' = F' \cup \{(map(end_{LIG}), p'')\}$;
31 $M' = (P', T', F', A, \ell \cup \ell_{LIG})$;
```

move-on-log is found, then the corresponding event in σ_e is marked as $start_{LIG}$; otherwise (i.e., if a move-on-model is found), it scans η forward until it finds a move corresponding to an event in σ_e (i.e., either a synchronous move or a move-on-log), that is then marked as $start_{LIG}$. The position of $start_{LIG}$ in σ_e is also returned ($posStart_{LIG}$). Finally, *findLinks* returns the transition of M corresponding to the last synchronous move before $start_{LIG}$ ($start_M$) and its position in σ_M. The function *findLinks* is called for finding connection points both at the beginning and at the end of LIG$_{\sigma_e}$ (lines 3 and 7, respectively). In the latter case, σ_e and a portion of the alignment are reversed and scrolled. In details, we consider the first part of the alignment through the first occurrence of σ_e (lines 5 and 6). This is done to properly find the end of the first occurrence of the extended embeddings. The function *left(seq, pos)* is used to return a subsequence of *seq* formed by its first *pos* elements. Additional steps are needed to determine $posEnd_M$ and $posEnd_{LIG}$ from the indexes obtained calling $findLinks$ on the reversed sequences (lines 8–9).

The places of the model to which to connect LIG$_{\sigma_e}$ correspond to the markings reached after executing the transition $start_M$ and before executing end_M, respectively. To find these places, the function *status* is used. It returns places of M with a token after the execution of the given sequence of transitions (lines 10 and 11). Specifically, the subsequences of σ_M starting from the beginning to $posStart_M$ and $posEnd_M - 1$ (recall that we want the input places to end_M) respectively.

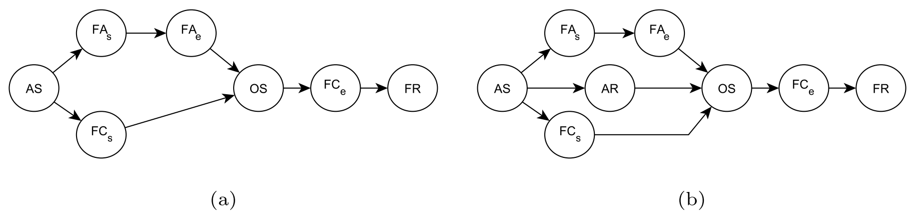


Fig. 9. (a) LIG$_{\sigma_{e_1}}$, (b) LIG$_{\sigma_{e_3}}$.

Algorithm 2: Auxiliary function: addStartEnd

```
   Input : The graph $LIG_{\sigma_e} = (E, W, l)$, and the extended
           embedding $\sigma_e$
   Output: The modified graph $LIG'$, the modified extended
           embedding $\sigma'_e$, the start node ($start$) and the end node
           ($end$) of $LIG'$
1  Function ($LIG'$,$\sigma'_e$,$start$,$end$)=addStartEnd((E,W,l),$\sigma_e$)
2      $Snode = \{e \in E | \nexists e' \in E, (e', e) \in W\}$;  /* set of start nodes */
3      $Enode = \{e \in E | \nexists e' \in E, (e, e') \in W\}$;  /* set of end nodes */
4      if $|Snode| > 1$ then
5          $E = E \cup \{H_S\}$;
6          $l = l \cup \{(H_S, hidden\}$;
7          $\forall e_i \in Snode, W = W \cup \{(H_S, e_i)\}$;
8          $\sigma_e = append(\langle H_S \rangle, \sigma_e)$;  /* append two sequences */
9          $start = H_S$;
10     else
11         $start = e, e \in Snode$;  /* LIG with unique start node */
12     if $|Enode| > 1$ then
13         $E = E \cup \{H_E\}$;
14         $l = l \cup \{(H_E, hidden\}$;
15         $\forall e_i \in Enode, W = W \cup \{(e_i, H_E)\}$;
16         $\sigma_e = append(\sigma_e, \langle H_E \rangle)$;
17         $end = H_E$;
18     else
19         $end = e, e \in Enode$;  /* LIG with unique end node */
20     return $(E, W, l), \sigma_e, start, end$;
```

Algorithm 3: Auxiliary function: findLinks

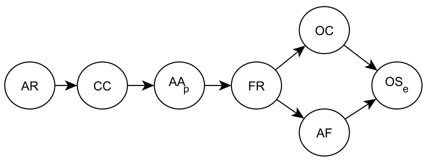
```
   Input : The extended embedding $\sigma_e$ of LIG in a trace $\sigma$, the
           alignment $\eta$ of $\sigma$
   Output: The event corresponding to the first non-synchronous
           move occurring in $\sigma_e$ ($link_{LIG}$), its position in $\sigma_e$
           ($posLink_{LIG}$), the transition of the model
           corresponding to the last synchronous move before
           ($link$), and its postion in $\sigma_M$ ($posLink_M$)
1  Function
   ($link_{LIG}$,$posLink_{LIG}$,$link_M$,$posLink_M$)=findLinks($\sigma_e$,$\eta$)
2      $i = j = k = 1$;
3      $done = True$; $flag = True$;
4      while $i < length(\eta) \wedge done$ do
5          $(t_i, a_i) = \eta[i]$; $v_i = act^{-1}(a_i)$;
6          if $t_i == \gg$ then  /* move-on-log */
7              if $v_i == \sigma_e[j]$ then
8                  $startNode = \sigma_e[j]$;
9                  $done = False$;
10         else
11             if $v_i \neq \gg$ then  /* synchronous move */
12                 if $flag$ then
13                     $posLastSync = k$;
14                     $lastSync = t_i$;
15                     if $v_i == \sigma_e[j]$ then
16                         $j = j + 1$;
17                 else
18                     $startNode = \sigma_e[j]$;
19                     $done = False$;
20             else if $j > 1$ then /* move-on-model within $\sigma_e$ */
21                 $flag = false$;
22             $k = k + 1$;  /* synchronous move or move-on-model */
23         $i = i + 1$;
24     return $startNode, j, lastSync, posLastSync$;
```

Now, in order to keep the model as simple as possible, LIG$_{\sigma_e}$ is modified by eliminating all nodes corresponding to transitions not to add to M. These are all events in σ_e corresponding to synchronous moves in η occurring before/after $posStart_{LIG}/posEnd_{LIG}$. We use E' to represent the set of nodes to add to the model, which are the nodes in E that correspond to one event in σ_e included in the interval between $posStart_{LIG}$ and $posEnd_{LIG}$ (line 12). Consequently, we keep only the arcs linking nodes in E' (line 13). If there are multiple start and/or end nodes after the transformation, then a new start and/or end node is added invoking again $addStartEnd$ (line 14). Note that if no modifications are made, then $start_{LIG}$ and end_{LIG} remain the same.

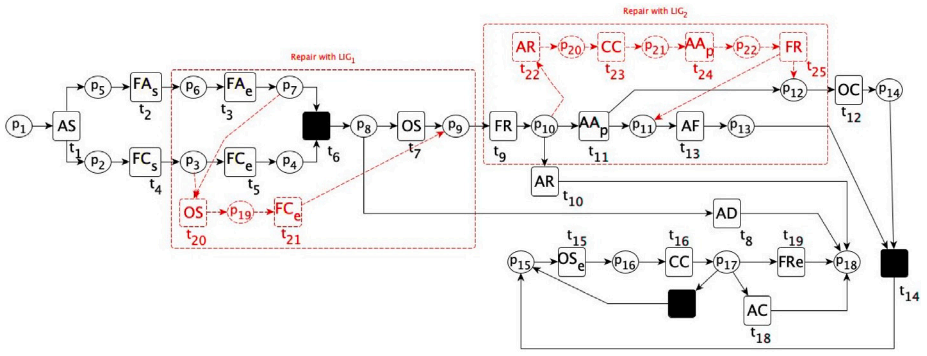
The resulting LIG$_{\sigma_e}$ is then transformed in a WF-net $M_{LIG} = (P_{LIG}, T_{LIG}, F_{LIG}, A, \ell_{LIG})$ (lines 15–25). To this end, we define the functions *create_transition* and *create_place* to generate a new transition and a new place, respectively. Every time that we create a transition, we update the mapping *map* between events in the graph and transitions in the set T_{LIG}, as well as the set T_{LIG} itself (lines 17–18). Furthermore, if the event does not correspond to a hidden transition (i.e, its label is not "hidden"), we update the labeling function ℓ_{LIG} too (line 20). Then, for each edge (e_i, e_j) in LIG$_{\sigma_e}$ a place p and the flows $(map(e_i), p)$ and $(p, map(e_j))$ are added to the Petri net (lines 22–25).

The places, transitions and flows obtained from the LIG are then added to the corresponding sets of the original model (line 26). Furthermore, for each place $p' \in P_S$ an edge $(p', map(start_{LIG}))$ is added to the new flow relation; similarly, a new edge $(map(end_{LIG}), p'')$ is added for each place $p'' \in P_E$ (lines 27–30). Finally, the repaired model M' is created (line 31).

Example 4. Let us consider again LIG$_1$, the trace σ_1 and the extended embedding σ_{e_1} introduced in Example 3. The alignment between the original process model and σ_1 is shown in Fig. 4. Let us apply Algorithm 1 to this example. The function $addStartEnd$ does not modify the LIG, since it already has a single start and a single end. Invoking the function $findLinks$, we will obtain $start_{LIG} = OS$, since it is the first event in σ_e corresponding in η to a synchronous move after the move-on-model corresponding to t_5. Its position in σ_e is $posStart_{LIG} = 5$. The transition corresponding to the last synchronous move before the move-on-model is $start_M = t_4$; the position of the corresponding event in



(a) LIG_2.



(b)

Fig. 10. (a) LIG corresponding to the anomalous behavior identified in σ_2, (b) the loan process model updated to include both LIG_1 and LIG_2.

σ_M is $posStart_M = 4$. Similarly, by invoking the function again to find the final connection points (lines 7–9), we obtain $end_{LIG} = FC_e$, since it is the last event in σ_e corresponding to a move-on-log, with its position $posEnd_{LIG} = 6$. Accordingly, $end_M = t_9$ and $posEnd_M = 8$. Therefore, we have $P_S = \{p_3, p_7\}$ and $P_E = \{p_9\}$. The next step is determining the elements of the LIG to be added to model. In our example, $E' = \{OS, FC_e\}$, since these are the only elements in σ_e in between $posStart_{LIG}$ and $posEnd_{LIG}$. Accordingly, $W' = \{(OS, FC_e)\}$ (line 13). The function $addStartEnd$ is called again, but with no effects, since the LIG already has single start/end nodes. Now we are ready to convert the LIG in a Petri net (lines 15–25); in particular, we will have $T_{LIG} = \{t_{20}, t_{21}\}$; $\ell_{LIG} = \{(t_{20}, OS), (t_{21}, FC)\}$; $P_{LIG} = \{p_{19}\}$; and $F_{LIG} = \{(t_{20}, p_{19}), (p_{19}, t_{21})\}$. We add the elements of the newly created Petri net to the corresponding sets of the original model M (line 26). Finally, we add an arc from p_3 and from p_7 to the transition OS (corresponding to the $start_{LIG}$ event), as well as an arc from FC (corresponding to the end_{LIG} event) to p_9 (lines 27–30). Fig. 10 shows the updated model M' (note that the model portion repaired according to LIG_1 is highlighted in the first dotted rectangle).

For the sake of space, we are not going to discuss the application of Algorithm 1 in detail for the second anomalous behavior (corresponding to σ_2) introduced in Section 1. Nevertheless, in Fig. 10 we report LIG_2 describing the behavior, together with the process model updated to include it.

If we compare the model repaired by using our approach with the one discussed in Section 1, it is straight to see that the model returned by our approach is able to represent the anomalous behaviors with a much higher precision than the competitor, even though at the cost of adding a higher number of duplicated transitions. A more elaborated discussion on the trade-off achieved by the two approaches will be presented when discussing the experimental results.

It is worth noting that the approach can deal also with complex synchronization constructs. As an example, let us suppose to add a small change to the process in Fig. 1 by introducing place p_9, which connects FA_s and FC_e. In this configuration, the fraud checking can be completed only if the first assessment of the application is marked at least as start. The black elements in Fig. 11 show the portion of the original model affected by the change (the black dashed elements stand for the remaining part of the model, not drawn here for the sake of simplicity). Now, let us assume to have a non-compliant trace $\sigma_4 = \langle AS, FC_s, FA_s, FA_e, CC, FR, OS, \ldots\rangle$. Here, the fraud checking did not end according to the prescribed course; instead, the customer has been contacted and the manager involved to take a decision on continuing the application procedure, which was positive. After this, the procedure comes back on the normal flow with OS. Let us assume that we extract a sequential LIG connecting the nodes FC_s, CC, FR; the red, dashed nodes in Fig. 11 show the model repaired to include this LIG. It should be noted that the starting marking for the LIG involves the places p_3, p_9, p_7, which guarantee that no dangling tokens are left even in the presence of this complex synchronization.

It can be easily demonstrated that the proposed repairing algorithm preserves the soundness of the model, as shown in the following theorem.

Theorem 1. *Let M be a WF-net, L a LIG, and M' the WF-net obtained by repairing M with L through the LIG integration Algorithm (Alg. 1). If M is sound, then M' is sound.*

Proof. To demonstrate the theorem, first of all let us observe that the WF-net M_{LIG}, generated from L, is integrated into M as an alternative path. In fact, recall from Alg. 1 that M_{LIG} is a WF-net with a unique (possibly hidden) initial transition t_i. A set of places P_S, corresponding to a reachable marking of M, is connected to t_i by adding an edge from each place $p \in P_S$ to it (lines 27–28 in Alg. 1). Therefore, from this marking the execution of t_i disables the execution of any other transition of M with the same precondition. Note that the marking is surely reachable since it corresponds to the set of places marked by firing a sequence of transitions in M. Similarly, the last transition of M_{LIG} is connected to a reachable marking of M (i.e. the set of places P_E). To demonstrate the soundness of M' it is thus sufficient to demonstrate the soundness of M_{LIG}. This directly comes from the observation that, since a LIG has only sequences and parallel branches, and no loops, then M_{LIG} is an acyclic Marked Graph. It is in fact known that acyclic Marked Graphs are always sound (a proof is reported in [27]).

5. Experiments

This section discusses the results of experiments performed to assess the proposed approach. We compare our results with the state-of-the-art technique in model repair [5], using the corresponding plug-in

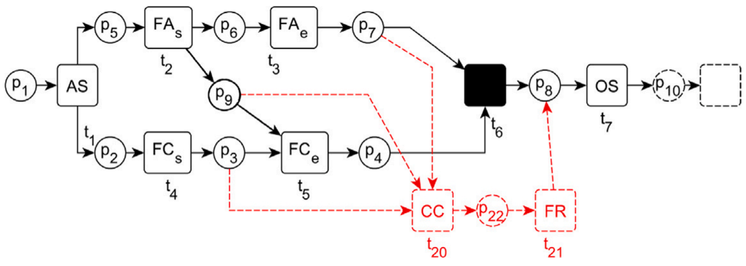


Fig. 11. Example of a complex parallelism construct.

available at the ProM github repository,[5] which is the official repository for the ProM platform [28]. We decided to compare our approach against [5] since it is the closest to ours in terms of repairing strategy among (semi)automatic process repair approaches. Other repairing techniques, and how they differ from our approach, are discussed in Section 2. The approach proposed in this work has been implemented as a Python script, available at the project GitHub repository[6] along with the used datasets.

5.1. Experiment settings

We carried out the evaluation in two scenarios, that we call global and local. The former is aimed at comparing the performance of the tested approaches while repairing exactly the same anomalies, while the second is devoted to cover the intended use of our repair approach. The introduction of a global scenario is because the technique in [5] does not allow one to select the anomalies to be incorporated in the model, but it repairs all the not synchronous moves in the alignment. As a consequence, in order to ensure that differences in the outcomes are only due to the repair strategy adopted to incorporate the anomalous behaviors in the model, we consider a synthetic log where an a-priory known set of anomalous behaviors was injected, without any additional noise. To ensure our approach realizes a complete repair, we selected the minimum set of anomalous LIGs covering all the injected behaviors, and we repair the original net incrementally, one LIG at a time. As regards the approach in [5], we simply use the entire event log as input.

In the local scenario, we want to assess the effect of incorporating just one selected LIG. To compare our results with the technique in [5], we gave as input to their plugin only the traces in the event log involving the LIG we decided to add, while the performance metrics (defined below) have been computed for the entire log. We hasten to note that in this case differences between the approaches will be due to both the repair strategy of the LIG and handling of noisy behaviors not belonging to the LIG.

For both scenarios, we tested the approach proposed in [5] by activating both the with and without loops detection option, which respectively takes/does not take into account the possible presence of loops during the repairing process, and with and without the post-processing options aimed at removing the less frequently used nodes. Similar post-processing optimizations may be in principle applied to our approach as well. We plan to introduce these optimizations in future work.

5.2. Performance metrics

We evaluate the quality of each repaired model along three performance metrics, commonly used when assessing process model quality: *fitness*, which calculates the amount of behaviors occurring in the event log that is also allowed by the process model [24]; *precision*, which assesses how much of the behavior allowed by the model actually occurs in the event log [29]; and the model *simplicity*, which evaluates the structural complexity of the model [30]. Each measure has been computed relying on the standard implementations provided by the PM4Py suite [31]. In particular, we used the alignment-based methods for the computation of fitness and precision, which have been assessed on the entire event log. As regards simplicity, the metric implemented in the PM4Py suite does not take into account the amount of duplicated transitions, while they are expected, in practice, to hamper the overall understandability of a model. Therefore, to provide a comprehensive assessment of the structure of the repaired model, we also report the overall number of places, transitions and arcs added by the repair.

We want to point out that, typically, there is a trade-off between fitness and precision, similar to the trade-off between precision and recall in information retrieval or the underfitting problem in classification. Furthermore, it is not always the case that process models with perfect fitness or precision properly represent the corresponding process. Classic examples of this issue are the so-called “flower models” [1], whose fitness is always equal to 1, since all possible sequences of activities are allowed; however, precision of such models is really poor and, furthermore, by allowing any activity execution order the model says little or nothing on the actual process behaviors. In this case the model is said to suffer from underfitting or, equivalently, to over-generalize. Elaborating upon these observations, the goal of the experiments is to assess the capability of the approaches to determine a good balance between fitness and precision. To this end, we also report the F1 measure, which is the harmonic mean of fitness and precision.

We conducted our analysis on two synthetic and three real-world datasets. The aim of the experiments on synthetic data is to perform controlled experiments and assess the effectiveness of the approach. We used real-life case studies to show that the approach provides useful insights and is robust to logs and models with real-life complexity.

5.3. Global repair experiments

5.3.1. Settings

For the global repair scenario, we used the process used for the experiments in [4], representing the checking of the profile of a receiver of a money transfer. We used the first dataset described in their synthetic experiments, where authors generated the event log by CPN Tools [32] inserting a number of anomalies, namely *swaps*, *repetitions* and *replacements*. Each anomaly has a probability of occurrence of 30%. Also note that it is possible that different anomalies appear in the same trace.

The first row of Table 1 shows some basic characteristics of the synthetic event log (note that the acronym “e.p.t.” stands for “events per trace”). In total eleven LIGs were extracted, to cover all anomalous behaviors.

5.3.2. Results

Table 2 reports the results obtained in terms of fitness, precision and simplicity on the models repaired by our approach, referred to as **LIG-based**, and by the approach in [5], referred to as **Low-level**, since it repairs sequences of low-level anomalies. Whether or not the loop

[5] https://github.com/promworkbench/ModelRepair.
[6] https://github.com/KDMG/process-mining/tree/master/ProcessRepairing.

Table 1
Event logs used for the synthetic and the real-world experiments.

Dataset	Traces	Events	Avg. e.p.t.	Min e.p.t.	Max e.p.t.
Synthetic No Noise	1500	30 238	20	13	41
Synthetic With Noise	1995	38 963	20	12	41
Fine Dataset	146 358	551 509	4	2	20
BPI2012	7455	85 426	11	4	82
BPI2017	3093	124.866	40	12	118

detection option was enabled for the low-level approach is indicated by the label +*L* or −*L*, respectively. Similarly, the label +*I* or −*I* stands for the settings with and without post-processing, respectively. As regards the fields *ΔArcs*, *ΔTrans* and *ΔPlaces*, we report the difference (both as absolute and percentage values) between the number of those elements before and after the repair. Percentage values represent the gain over the number of elements in the original model, where 0% means than the number of elements does not change, while 100% means that the repaired model has twice as many elements as the original model. For reference purposes, we also report the values of the original net, before the repairing. Note that, in this case, the *ΔArcs*, *ΔTrans* and *ΔPlaces* fields have no meaning (they would correspond to the difference between the original net and itself, so they would always be zero), so they are not reported.

Both approaches were able to incorporate all anomalies into the original model, enhancing the fitness up to 1, as expected. However, the other metrics, in particular precision, show notable differences. While all the repaired models show a worsening in precision, the LIG-based approach returns the highest precision value. In contrast, the low-level approach decreases precision to 0.32 or 0.31 depending on the different loop and post-processing settings. We do not report the F1 measure in the table for formatting reasons, observing that being fitness equal to 1 for all repaired models the measure is governed by precision. However, it can be interesting to note that the LIG-based approach achieves an F1 score which is almost the same of the original model (0.888 vs. 0.893). These results confirms what was already apparent when comparing the example in Figs. 1 and 10b. The results also show that the LIG-based approach returns a simpler model than the competitor. Indeed, while they score similarly in terms of the simplicity metric (the difference between the approaches is between 0.03 and 0.05), low-level approaches add more elements to the model: Low-Level +L added 31.3% more elements (arcs, transitions and places) than the LIG-based, whilst Low-level −L adds 48.4% more elements. It is worth noting another characteristic of the approach. The number of LIGs produced at the end of the anomalous subgraph extraction phase is 191 on this dataset. This is because the LIGs can include parts of the same anomaly injected in the log combined with other compliant parts of the behavior. The procedure of LIG integration, discarding these compliant parts, can be iterated to include all the anomalies without introducing redundancies, as the values of precision and simplicity demonstrate.

To qualitatively evaluate the effect of the two different repair strategies, let us consider the repair of a single (structured) anomaly. Fig. 12 shows the original process model. The part framed in red describes a pre-profiling phase. For this phase an anomaly has been injected in the log, which swaps SRPP and FRPP activities. Figs. 13 and 14 show the repair of this anomaly performed by the LIG-based approach and by [5] respectively. We can see in Fig. 13 the added alternative path in red, with the rest of the net unchanged. Similarly, in Fig. 14 in red the corresponding repair. It is apparent the “structuredness” of the LIG-based approach contrasted with the independent management, by [5], of asynchronous moves resulting from the SRPP/FRPP swap. As a result, the anomalous behavior is now allowed: by following hidden activities we can now skip SRPP, RBPC, REPC, and RIBPC, enabling FRPP soon after SRP. Then, a loop allows to execute RBPC, REPC, and RIBPC, and finally SRPP. Of course, many other paths are now allowed, e.g. skipping SRPP and executing RBPC, REPC, and RIBPC in parallel soon after SRP, which is not an expected behavior. The example allows us to also enlighten the differences between the structures used for repair in the two approaches: (1) from the definition of LIG derives that it will not contain neither choices nor loops, while these control flow structures can appear in the models used for repair by [5] since they are synthesized by process discovery on sublogs. (2) Since sublogs are composed of maximal sequences of anomalies sharing the same starting point (called location by the authors) they do not include synchronous moves, whilst LIGs can. As a consequence, even a single synchronous move in between two anomalous sequences leads [5] to recognize two distinct substructures, while our approach can produce a single bigger anomalous structure. This feature leads LIG-based approach to repair with greater precision and to the introduction of duplicate transitions.

In Fig. 14 we can also note other changes operated by [5] to the original model, enlightened in cyan. These are the repair of other anomalies injected into the log. This enlighten a further difference between the two repair approaches which demonstrate an inherent difficulty in comparison, and the consequent motivation behind the introduction of the global repair scenario: as a matter of fact, by controlling and repairing all the existing anomalies, differences in performances are only due to the different repair strategy, and not to different anomalies repaired. As to the LIG-based approach, we highlight that by inserting a frequent LIG, representative of the anomalous behavior in a set of traces instead of the whole traces, we only insert the desired behavior and control the over-fitting problem appropriately. Finally, we note that in this particular example where the start and end nodes of the LIG correspond to the swap nodes (to the anomalous behavior) the whole LIG is entered as an alternative path. However, in the general case, where the anomalous behavior is contained in a larger LIG, the synchronous parts of the LIG are not added to the model thus limiting the redundancy of the nodes, as shown in Fig. 10(b).

5.4. *Local repair experiments*

5.4.1. *Settings*

For this scenario, we used two synthetic and three real-world datasets. For the synthetic experiments, we used both the log discussed in the previous section, and a second, larger, synthetic log describing the same process but with higher percentage of noise. In particular, in addition to the anomalies previously mentioned, process activities were randomly added/removed in portions of the process not involved by the occurrence of the anomalies introduced for the previous experiment. The amount of noise added to a trace is equal to the 10% of the length of the trace. This creates quite a challenging scenario, which allows us to test the robustness of the approach to high levels of noise. The second line of Table 1 reports basic statistics on this event log (i.e., Synthetic - With noise). In addition, we used three publicly available event logs. The first and the second one, hereby referred to as BPI2012 [33] and BPI2017 [34], both record the loan management process of a Dutch Financial Institute, made available for the BPI 2012 and 2017 BPI challenges. These event logs contain detailed information about applications submitted by clients, loan offers sent by the company, and work items processed by employees or by the system. It should be noted that this process went through important changes between 2012 and 2017 (among which, the implementation of a new workflow system), with the result that the two logs actually describe two different processes. We applied the same preprocessing (e.g., removing incomplete process executions) and used the same process models derived for these event logs in [4], reported in their real-world experiments section. The third log describes the management of road traffic fines by a local police force in Italy [35]. The event log contains information about the fine and different possible handling paths (e.g., the offender could pay as soon as notified or go through a court to appeal). Since the longest case we observed in the log took almost one year, we removed cases that started after the 30th of June 2012 (i.e., with less than one year left till the end of the logging period) as a heuristic to filter out

Table 2
Results for the global repair experiments. Best results are in boldface.

Dataset	Net	Fit.	Prec.	Simp.	ΔArcs	ΔTrans	ΔPlaces
Synthetic No noise	Original	0.85	0.94	0.76	–	–	–
	LIG-based	**1.00**	**0.80**	**0.58**	**41 (70.7%)**	**14 (58.3%)**	9 (34.6%)
	Low-level +L	**1.00**	0.32	0.54	55 (94.9%)	23 (95.8%)	6 (23.1%)
	Low-level −L	**1.00**	0.31	0.54	61 (100%)	26 (108.3%)	8 (30.8%)
	Low-level +L+I	**1.00**	0.32	0.53	47 (81%)	19 (79%)	**4 (15%)**
	Low-level −L+I	**1.00**	0.32	0.55	51 (88%)	21 (88%)	6 (23%)

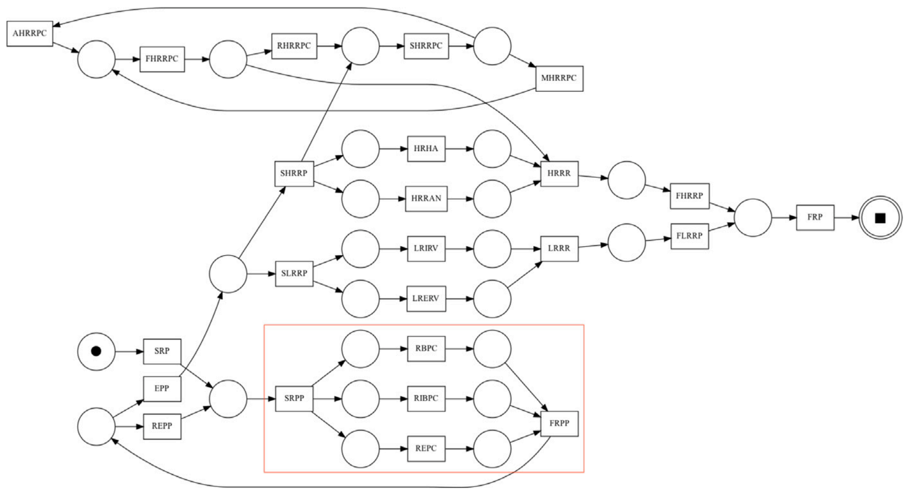


Fig. 12. Money transfer process model with the position of a synthetic anomaly (framed in red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

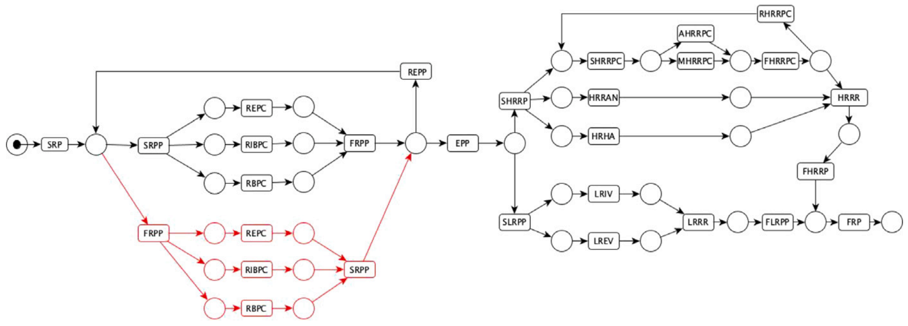


Fig. 13. Repair of the anomaly with the LIG-based approach on the money transfer process (in red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

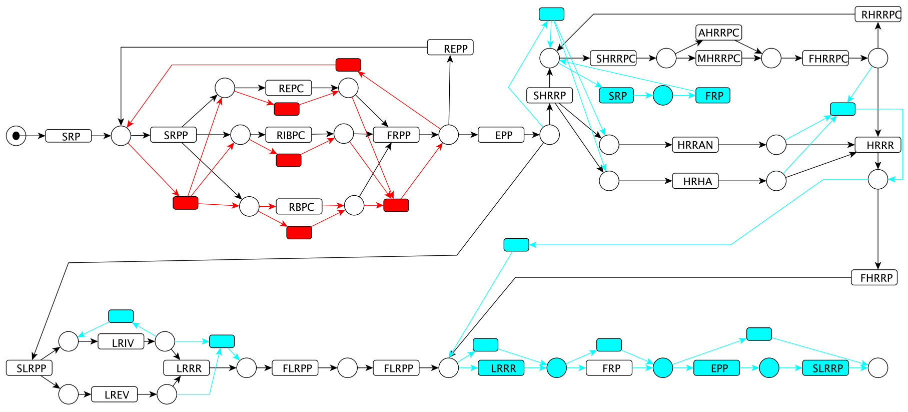


Fig. 14. Repair of the anomaly performed by [5] on the money transfer process (in red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Results for local repair experiments: fitness, precision and F1 measure. Best results are in boldface.

Dataset	Net	Fit.	Prec.	F1
Synthetic No Noise	Original	0.85	0.94	0.89
	LIG-based	0.89	**0.92**	**0.90**
	Low-level +L	**0.95**	0.38	0.54
	Low-level −L	**0.95**	0.34	0.50
	Low-level +L+I	**0.95**	0.39	0.55
	Low-level −L+I	**0.95**	0.39	0.55
Synthetic With Noise	Original	0.85	0.96	0.90
	LIG-based	0.86	**0.96**	**0.91**
	Low-level +L	**0.96**	0.37	0.53
	Low-level −L	**0.96**	0.40	0.56
	Low-level +L+I	0.95[a]	0.39[a]	0.55
	Low-level −L+I	0.95[a]	0.41[a]	0.57
Fine	Original	0.996	0.88	**0.93**
	LIG-based	0.997	**0.88**	**0.93**
	Low-level +L	**0.999**	0.77	0.87
	Low-level −L	**0.999**	0.77	0.87
	Low-level +L+I	**0.999**	0.75	0.86
	Low-level −L+I	0.998	0.81	0.89
BPI2012	Original	0.92	0.49	**0.64**
	LIG-based	0.93	**0.48**	0.63
	Low-level +L	**0.98**	0.28	0.44
	Low-level −L	**0.98**	0.28	0.44
	Low-level +L+I	**0.98**	0.29	0.45
	Low-level −L+I	**0.98**	0.28	0.44
BPI2017	Original	0.81	0.78	**0.79**
	LIG-based	0.83	**0.72**	0.77
	Low-level +L	0.88	0.50[a]	0.64
	Low-level −L	**0.98**	0.13	0.23
	Low-level +L+I	0.87	0.38	0.53
	Low-level −L+I	**0.98**	0.12[a]	0.21

[a] Denotes averages computed on a subset of LIGs.

incomplete cases. For the process model, we used the one built by previous research [36]. It has been manually designed taking domain knowledge and regulations about the management of road traffic into account. The last three rows of Table 1 show some basic characteristics of the real-world event logs.

For each dataset, the number of LIGs found after the anomalous subgraph extraction phase is: 191 (Synthetic No Noise), 485 (Synthetic With Noise), 181 (Fine), 60 (BPI2012), and 35 (BPI2017). We selected the 10 most relevant LIGs to apply our repair algorithm (i.e., the LIGs with the highest Description Length), and we assessed the quality of the repaired model separately for each LIG, averaging the results. On average, the size of the LIGs in terms of number of nodes is: 11.6 (Synthetic No Noise), 6.6 (Synthetic With Noise), 3.7 (Fine), 6.8 (BPI2012), and 9.2 (BPI2017).

5.4.2. Results

For the sake of space, the results are split into two tables. Table 3 shows the results related to the fitness and precision metrics, while Table 4 shows the results related to the simplicity related metrics. For each approach and metric we report the average of the values over all LIGs. As regards the fields *ΔArcs*, *ΔTrans* and *ΔPlaces*, we report the average difference between the number of each of those elements before and after the repair.

In the first synthetic dataset, loop detection and post-processing do not affect significantly the performance of the low-level repair, which obtained in all configurations the best results in terms of fitness (0.95 against 0.89 of the LIG-based approach). However, when it comes to the precision, it is straight to see that the LIG-based approach outperforms the low-level repair. Indeed, the highest average precision obtained by low-level repair is 0.39, against the 0.92 returned by the LIG-based repair. A similar trend can be observed for simplicity: the LIG-approach scored better than low-level repair techniques (0.72 against values in [0.58; 0.60]). Another interesting observation is that our approach added considerably less elements to the repaired net than the low-level one, as shown by the number of added arcs, transitions and places.

Differences between the tested approaches are even more evident in the remaining datasets, likely because of the higher level of noise. Overall, we can observe that our approach achieved an almost negligible improvement of the fitness. This result is in line with the expectations; we selected datasets with a very high level of noise (with the exception of the Fine dataset, discussed later in the text), from which we only repair a portion of the most frequent anomalous behaviors, which explains why the fitness did not improve that much. Nevertheless, the low-level approach managed to improve the fitness of the original models within 6% (for the BPI2012 dataset) and 17% (for the BPI2017 dataset)

However it is straight to see that such an improvement of fitness has been obtained at the expenses of a strong decrease in quality for the other metrics. Indeed, for all the datasets we observe a drop in precision, particularly evident for results obtained by the low-level without loops on the BPI2017 dataset, where we observe a drop from 0.78 to 0.12/0.13. Regarding this dataset, it should be noted that for the low-level with loops we were able to compute the precision values only for five of the LIGs; for the others, it was not possible to conclude the computation because of an out of memory error (on a 64 GB RAM machine). Therefore, the 0.5 precision value for the low-level +L configuration is only a partial value. Similarly, it was not possible to compute metrics of some LIGs with the post-processing option enabled on the Synthetic and BPI2017 datasets. These case are enlightened in Table 3 with an asterisk.

Indeed, from the definition, low precision values indicate that more variants are possible in the model than in the log. Hence very low precision values are indicative of a model with potential over-generalization issues. As observed before, the starting experimental conditions have been made as similar as possible but they are not identical for the two approaches, since the LIG-based repairs a specific anomaly, while the low-level is provided with the subset of traces that contain the same anomaly, possibly together with others. Thus higher fitness and lower precision is expected. However, as F1 scores demonstrate, fitness increase is not balanced by the deterioration of precision, and the LIG-based approach is able to provide an overall better balance. Similar observations, advocating for local, curated repairs, are discussed in [8].

Another interesting trend that emerges from Table 4, is that the low-level repair approach seems to return more complex models than the LIG-based approach. The LIG-based approach scores consistently better in terms of simplicity; furthermore, the low-level approach adds in general far more arcs and transitions, up to one order of magnitude.

It is worth noting that the Fine dataset shows different initial conditions; indeed, the original net already presented almost a perfect fitness, showing that few anomalous behaviors occurred in the dataset. Therefore, there is not much room for improvement for this metric; indeed, improvements achieved by both the approaches are only noticeable to the the third decimal, that is the reason why we reported three decimals for this value in Table 3. However, it is interesting to observe that the considerations already made on the precision and the simplicity of the approaches continue to hold; namely, the low-level repair led to a noticeable worsening of both precision and simplicity (decreased of 6.2% and 1.7% w.r.t. the original model when only the post-processing option is enabled, respectively), whilst the LIG-based approach kept values close to the original model. Furthermore, the low-level repair added on average much more transitions and arcs than the LIG-based. These results are interesting, since they highlight that even in conditions of low noise, low-level repair can lead to imprecise and complex models.

Table 4
Results for local repair experiments: simplicity. Best results are in boldface.

Dataset	Net	Simp.	ΔArcs	ΔTrans	ΔPlaces
Synthetic No Noise	Original	0.76	–	–	–
	LIG-based	**0.72**	**8.60 (14.8%)**	**3.00 (12.5%)**	2.80 (10.8%)
	Low-level +L	0.58	32.00 (55.2%)	13.10 (54.6%)	2.60 (10.0%)
	Low-level –L	0.60	35.00 (60.3%)	14.60 (60.8%)	3.70 (14.2%)
	Low-level +L+I	0.58	28.80 (49.7%)	11.50 (47.9%)	**1.80 (6.9%)**
	Low-level –L+I	0.59	31.00 (53.4%)	12.60 (52.5%)	2.90 (11.1%)
Synthetic With Noise	Original	0.75	–	–	–
	LIG-based	**0.72**	**9.50 (16.4%)**	**3.40 (14.2%)**	**2.60 (10.0%)**
	Low-level +L	0.46	191.40 (330.0%)	87.10 (363.0%)	11.10 (42.7%)
	Low-level –L	0.47	228.10 (393.3%)	104.00 (433.3%)	14.80 (56.9%)
	Low-level +L+I	0.47	142.00 (244.8%)	65.00 (270.8%)	8.11 (31.2%)
	Low-level –L+I	0.44	216.30 (372.9%)	97.60 (406.7%)	14.40 (55.4%)
Fine	Original	0.58	–	–	–
	LIG-based	**0.58**	**2.80 (7.0%)**	**1.40 (7.0%)**	0.40 (4.4%)
	Low-level +L	0.52	11.00 (28.0%)	5.50 (29.9%)	**0.00**
	Low-level –L	0.54	7.80 (20.5%)	3.90 (20.5%)	**0.00**
	Low-level +L+I	0.54	13.60 (35.8%)	6.80 (35.8%)	1.5 (13.3%)
	Low-level –L+I	0.55	10.60 (27.9%)	5.30 (27.9%)	1.2 (13.3%)
BPI2012	Original	0.69	–	–	–
	LIG-based	**0.68**	**8.70 (12.4%)**	**4.10 (12.4%)**	2.20 (1.0%)
	Low-level +L	0.62	13.80 (19.7%)	6.90 (21.0%)	**0.00**
	Low-level –L	0.62	13.80 (19.7%)	6.90 (21.0%)	**0.00**
	Low-level +L+I	0.62	13.80 (19.7%)	6.90 (21.0%)	**0.00**
	Low-level –L+I	0.62	13.80 (19.7%)	6.90 (21.0%)	**0.00**
BPI2017	Original	0.73	–	–	–
	LIG-based	**0.71**	**23.70 (13.0%)**	**10.80 (12.6%)**	6.70 (10%)
	Low-level +L	0.59	84.70 (46.5%)	38.70 (45.0%)	**3.20 (4%)**
	Low-level –L	0.61	143.00 (78.6%)	69.90 (81.3%)	21.90 (32.7%)
	Low-level +L+I	0.67	37.67 (20.7%)	18.33 (21.3%)	3.56 (5.3%)
	Low-level –L+I	0.61	128.00 (70.3%)	62.00 (72.1%)	19.33 (28.9%)

5.5. *Discussion*

Experimental results show that the LIG-based approach in general outperforms the low-level repair in terms of precision and simplicity, at the cost of an often modest improvement of fitness. This trade-off between fitness and precision can easily be explained by considering the characteristics of the tested logs, and reflects the main differences in terms of the adopted repair strategies. In the tested logs, the anomalous behaviors selected for the repair often co-occur with other low-level behaviors in the traces. The LIG-based repair is tailored to include specific structured anomalous behaviors, thus ignoring possible other anomalous behaviors in a trace. Therefore, we do not expect huge improvements in terms of fitness, since anomalous behaviors are often not very frequent in the event log, and we repair one of them at a time. The low-level approach [5], instead, does not allow the user to select the behaviors to repair, but it strives to repair all anomalous behaviors observed in the log. This explains the higher fitness. However, this is obtained at the expenses of a much lower precision These effects are more visible in the real-world datasets; in particular, as expected, with the increasing of the level of noise the drop in precision and simplicity increases as well, since the repaired models involve much more noisy behaviors than the synthetic one. Although global and local approaches respond to different goals and are not easily comparable, the F1 scores demonstrate the capability of the LIG-based approach to achieve a good balance between fitness and precision. As a further observation, we note that in principle a low-level approach can be adopted in a local setting. Similarly to what has been done in [8] a possible solution is to repair the event log, removing all anomalous behaviors which are not instances of the selected one. However, such pre-processing would not be enough to guarantee significant improvements in terms of precision: as shown by the example in Section 1.1 and the result of the experiments reported in Fig. 14, the low-level repair would anyway introduce more behaviors than intended, since co-occurrences among low-level anomalies, other than sequences, are not taken into account. This claim is also supported by the results obtained in the first set of experiments (Section 5.4), i.e., the global repair scenario, where the two approaches repaired exactly the same set of anomalous behaviors. In this case, the LIG-based approach performed much better in terms of precision, while achieving the same fitness of the low-level approach. In this regards, it is also worth noting that while the precision of the LIG-repaired model is less than the precision of the original model, the F1 score of the two turns out to be practically the same.

Similar considerations hold for the simplicity. However, while for precision we expect to perform in general better than the low-level approach, as explained above, the better performance of the LIG-based approach in terms of simplicity when no additional noise occurs are more dependent on the characteristics of the LIGs. The better performance of the LIG-based in the tested synthetic dataset are due to the fact that many of the LIGs started/ended with compliant behaviors, which were hence not added to the model; while the low-level repair added a number of additional nodes, e.g., hidden transitions. The presence of compliant activities in many LIGs is expected, because of their higher frequency in the log with respect to anomalous activities. Anyway, in the worst case of a LIG including only non-compliant activities, all nodes would be added to the process model, thus worsening the simplicity. In this respect, it is worth noting that a trade-off exists between understandability/readability, amount of duplicated transitions and amount of unwanted behavior in the repaired model. When duplicating transitions, one can hypothetically create a separate branch for every execution trace that exists in the log. That way, one can ensure that all behavior is captured and no other non-existing behavior is allowed. It is clear that such a model quickly becomes impossible to read and understand. On the other hand, creating a model that precisely specifies the wanted behavior without duplicating transitions could lead to very complicated models with many hidden transitions. The current approach relies on the addition of the minimum amount of duplicated transitions needed to incorporate the desired LIG. We observe that the percentages of duplicated transitions with respect to the size of the LIGs are: 29.9% (Synthetic No Noise), 48.6% (Synthetic With Noise), 83.6% (Fine), 42.5% (BPI2012), and 70.3% (BPI2017). This leads to a percentage of duplicated transitions in the repaired model ranging from 6.1% for BPI2017, to 13.2% for Fine. Other values are 13.0% (Synthetic No Noise), 11.8% (Synthetic With Noise), 7.5% (BPI2012). Although we consider these numbers a good result, we plan to delve into the possible use of hidden transitions, and how to achieve a better balance within these two alternative solutions in future work. Another drawback of alternative paths with duplicated transitions is that it can generate an overfitting model. This is especially risky when every non-compliant trace leads to an alternative path in the net. As to this concern we like to note that alternative paths represent anomalous behaviors that are *common* to a *set of traces*. The discovery of common anomalous patterns in the log reduces the risk of overfitting at some extent. Furthermore, the Minimum Description Length criterium adopted by SUBDUE to discover anomalous patterns guarantees that the largest frequent common patterns are found first, contributing to the generalization ability. The first structures found by SUBDUE should then be applied for repairing as a good practice to control overfitting, as we did in the experiments, although we do not impose this as part of the methodology since users could have repair goals other than frequency (e.g., if an anomaly related to a special treatment for few platinum clients leads to a substantial improvement of return, it can be justifiable to promote it to standard practice.). The evaluation of the generalization of the method by the PM4Py suite demonstrates good performance on the selected datasets: 0.96 for Synthetic No Noise, 0.97 for Synthetic With Noise, 0.93 for Fine, 0.86 for BPI2012, and 0.87 for BPI2017.

In terms of execution time, Anomalous LIGs Extraction represents the most expensive step of the methodology. In particular, the cost of IG building depends on finding the optimal alignment of all log traces; while the time for the application of repair rules is negligible. The execution of SUBDUE for extracting relevant subgraphs has a complexity that depends on the size of the IGs and the number of subgraphs to be extracted. In fact, at each iteration SUBDUE uses the subgraph with the highest DL to compress IGs, and use them as input for the next iteration. So the execution time depends on the number of relevant LIGs to extract. In the case of all anomalies in real-world should be repaired, SUBDUE should iterate until the first k most relevant LIGs covering all traces in the log have been discovered. Finally, the complexity of the LIG Integration step depends on the length of the alignment, which has already been defined during the building IG step. Since the approach in [5] is also based on optimal alignment, the difference stems in the way models for the repair are extracted: SUBDUE and process discovery respectively.

Finally, we would also like to point out that our approach is not constrained to the use of a specific LIG extraction methodology. Indeed, the only requirement to apply the proposed integration approach is to provide in input (a) a LIG fulfilling the properties described in Section 3 (which can also be manually drawn by the user), and (b) an event log involving at least one trace where the LIG occurs (which can also be simulated).

6. Conclusions

In this work, we introduced a process repair technique based on the use of local models representing high-level anomalous behaviors. These high-level behaviors correspond to frequently co-occurring low-level anomalous behaviors. The proposed approach allows to repair a single high-level anomalous behavior. The integration of multiple anomalies is still possible by iterating the procedure. We compared the approach with respect to state-of-the-art competitor, both on synthetic and real-world datasets, considering both a global and a local repair scenario. In the first one, all anomalies have been repaired to obtain a single repaired model, and differences in the output are only due to the repair strategies; in the second scenario, we evaluate the results obtained when incorporating only local behaviors, where also additional noise can occur, potentially impacting the obtained model. The results show that the proposed approach outperformed the competitor in terms of precision and simplicity of the repaired models, at the cost of an often modest improvement of fitness, showing a favorable balance in terms of F1 measure.

For future work, first, we intend to extend the approach to deal with multiple occurrences of a LIG within the same process execution. Another extension we plan to include, is to consider small variants of the identified LIGs, to improve fitness. Furthermore, we plan to investigate strategies, similar to loop detection and post-processing options in [5], to minimize the number of elements included into the model, improving the simplicity of the outcome. With the same aim, we plan to delve into the possible use of hidden, instead of duplicate, transitions and how to achieve a better balance within these two alternative solutions.

CRediT authorship contribution statement

Laura Genga: Writing – review & editing, Writing – original draft, Supervision, Software, Methodology, Formal analysis, Conceptualization. **Fabio Rossi:** Visualization, Software. **Claudia Diamantini:** Writing – review & editing, Writing – original draft, Supervision, Software, Methodology, Formal analysis, Conceptualization. **Emanuele Storti:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Formal analysis, Conceptualization. **Domenico Potena:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have used publicly available datasets. The link can be found in the paper.

References

[1] W.M.P. van der Aalst, Process Mining - Data Science in Action, second ed., Springer, 2016.

[2] A.A. Cervantes, N.R. van Beest, M. La Rosa, M. Dumas, L. García-Bañuelos, Interactive and incremental business process model repair, in: OTM Confederated International Conferences“ on the Move to Meaningful Internet Systems”, Springer, 2017, pp. 53–74.

[3] A. Adriansyah, J.M. Buijs, Mining Process Performance from Event Logs: The BPI Challenge 2012, BPM Center Report BPM-12-15, BPMcenter.org, 2012.

[4] L. Genga, M. Alizadeh, D. Potena, C. Diamantini, N. Zannone, Discovering anomalous frequent patterns from partially ordered event logs, J. Intell. Inf. Syst. (2018) 1–44.

[5] D. Fahland, W.M.P. van der Aalst, Model repair - aligning process models to reality, Inf. Syst. 47 (2015) 220–243.

[6] C. Diamantini, L. Genga, D. Potena, W.M.P. van der Aalst, Building instance graphs for highly variable processes, Expert Syst. Appl. 59 (2016) 101–118.

[7] A. Polyvyanyy, W.M.P. Van der Aalst, A.H.M. Ter Hofstede, M.T. Wynn, Impact-driven process model repair, ACM Trans. Softw. Eng. Methodol. 25 (4) (2016) 1–60.

[8] M. Dees, M. de Leoni, F. Mannhardt, Enhancing process models to improve business performance: A methodology and case studies, in: OTM Confederated International Conferences“ on the Move to Meaningful Internet Systems”, Springer, 2017, pp. 232–251.

[9] A. Adriansyah, B.F. Van Dongen, N. Zannone, Controlling break-the-glass through alignment, in: 2013 International Conference on Social Computing, IEEE, 2013, pp. 606–611.

[10] K. Revoredo, On the use of domain knowledge for process model repair, Softw. Syst. Model. (2022) 1–13.

[11] X. Zhang, Y. Du, L. Qi, H. Sun, An approach for repairing process models based on logic Petri nets, IEEE Access 6 (2018) 29926–29939.

[12] Y. Teng, Y. Du, L. Qi, W. Luan, L. Wang, A simple logic transition repair method for business process models via logic petri nets, IEEE Access 7 (2019) 76628–76644.

[13] J.C.A.M. Buijs, M. La Rosa, H.A. Reijers, B.F. Dongen, W.M.P. van der Aalst, Improving business process models using observed behavior, in: International Symposium on Data-Driven Process Discovery and Analysis, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 44–59.
[14] A.A. Mitsyuk, I.A. Lomazova, I.S. Shugurov, W.M.P. van der Aalst, Process model repair by detecting unfitting fragments, in: AIST (Supplement), 2017, pp. 301–313.
[15] C. Li, M. Reichert, A. Wombacher, Discovering reference models by mining process variants using a heuristic approach, in: Proceedings of the 7th International Conference on Business Process Management, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 344–362.
[16] D. Fahland, W.M.P. van der Aalst, Simplifying discovered process models in a controlled manner, Inf. Syst. 38 (4) (2013) 585–605, Special section on BPM 2011 conference.
[17] A. Kalenkova, J. Carmona, A. Polyvyanyy, M. La Rosa, Automated repair of process models using non-local constraints, in: International Conference on Applications and Theory of Petri Nets and Concurrency, Springer, 2020, pp. 280–300.
[18] M. Reichert, P. Dadam, ADEPT flex—supporting dynamic changes of workflows without losing control, J. Intell. Inf. Syst. 10 (2) (1998) 93–129.
[19] M. Gambini, M. La Rosa, S. Migliorini, A.H. Ter Hofstede, Automated error correction of business process models, in: International Conference on Business Process Management, Springer, 2011, pp. 148–165.
[20] N. Lohmann, Correcting deadlocking service choreographies using a simulation-based graph edit distance, in: Business Process Management: 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 132–147.
[21] W.M.P. van der Aalst, W.Z. Low, M.T. Wynn, A.H.M. ter Hofstede, Change your history: Learning from event logs to improve processes, in: 2015 IEEE 19th International Conference on Computer Supported Cooperative Work in Design, CSCWD, 2015, pp. 7–12.
[22] J. Ghattas, P. Soffer, M. Peleg, Improving business process decision making based on past experience, Decis. Support Syst. 59 (2014) 93–107.
[23] W.M.P. van der Aalst, Verification of workflow nets, in: P. Azéma, G. Balbo (Eds.), Application and Theory of Petri Nets 1997, Springer Berlin Heidelberg, 1997, pp. 407–426.
[24] A. Adriansyah, B.F. van Dongen, W.M.P. van der Aalst, Conformance checking using cost-based fitness analysis, in: 2011 Ieee 15th International Enterprise Distributed Object Computing Conference, IEEE, 2011, pp. 55–64.
[25] D.J. Cook, L.B. Holder, Substructure discovery using minimum description length and background knowledge, J. Artificial Intelligence Res. 1 (1993) 231–255.
[26] I. Jonyer, D.J. Cook, L.B. Holder, Graph-based hierarchical conceptual clustering, J. Mach. Learn. Res. 2 (Oct) (2001) 19–43.
[27] K. Van Hee, N. Sidorova, M. Voorhoeve, Soundness and separability of workflow nets in the stepwise refinement approach, in: International Conference on Application and Theory of Petri Nets, Springer, 2003, pp. 337–356.
[28] ProM, 2023, http://dx.doi.org/10.1007/11494744_25, Accessed: 2023-12-15.
[29] A. Adriansyah, J. Munoz-Gama, J. Carmona, B.F. Van Dongen, W.M.P. Van Der Aalst, Measuring precision of modeled behavior, Inf. Syst. E-Bus. Manage. 13 (1) (2015) 37–67.
[30] F.R. Blum, Metrics in Process Discovery, Tech. Rep., Technical Report, TR/DCC, 2015, pp. 1–21.
[31] PM4Py, 2023, https://doi.org/10.1016/j.simpa.2023.100556, Accessed: 2023-12-15.
[32] CPN Tools, 2023, https://doi.org/10.1007/s10009-007-0038-x, Accessed: 2023-12-15.
[33] B. van Dongen, BPI Challenge 2012. Version 1. 4TU.ResearchData. dataset, 2012, https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f, Accessed: 2023-12-15.
[34] B. van Dongen, BPI Challenge 2017. Version 1. 4TU.ResearchData. dataset, 2017, https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b, Accessed: 2023-12-15.
[35] M. de Leoni, F. Mannhardt, Road Traffic Fine Management Process. Version 1. 4TU.ResearchData. dataset, 2015, https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5, Accessed: 2023-12-15.
[36] F. Mannhardt, M. De Leoni, H.A. Reijers, W.M.P. Van Der Aalst, Balanced multi-perspective checking of process conformance, Computing 98 (4) (2016) 407–437.