

ReLIGn: a Tool for Model Repair based on Local Instance Graphs

Claudia Diamantini^{1,†}, Laura Genga^{2,†}, Chiara Gobbi^{1,*,†}, Alessandro Mele^{1,†} and Domenico Potena^{1,†}

¹Dipartimento di Ingegneria dell'Informazione, Università Politecnica delle Marche, Ancona, Italy

²Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract

Model repair techniques update process models to incorporate behaviors observed in event logs, but not compliant with the original model. While these techniques address important practical needs, most state-of-the-art approaches repair anomalous behaviors independently, neglecting potential correlations among anomalies occurring at different process stages. This limitation introduces potential issues, e.g., over-permissive and low-quality models. In this paper, we present ReLIGn, a novel tool for process model repair that includes in the original model a high-level anomalous behavior (AB) represented as a Local Instance Graph (LIG). The tool supports the user in the evaluation of the repaired model, both graphically by highlighting the repaired part of the model and numerically by reporting the difference in terms of fitness, precision, generalization and simplicity between the original and repaired model.

Keywords

Process Mining, Process Repairing, Process Model, Business Process Management

Metadata description	Value
Tool name	ReLIGn
Current version	0.1.0
Legal code license	GPL-3.0
Languages, tools and services used	Graphviz, Java, MySQL, Python
Supported operating environment	GNU/Linux, macOS, Microsoft Windows
Source code repository	https://github.com/KDMG/ReLIGn-tool
Screencast video	https://youtu.be/wafwsUPTees

1. Introduction

Model repair techniques aim at automatically updating a process model to incorporate behaviors that are observed in practice (i.e., in *event logs* tracking process executions) but are not compliant

ICPM Doctoral Consortium and Demo Track 2025, October 20-24, 2025, Montevideo, Uruguay

*Corresponding author.

[†]These authors contributed equally.

✉ c.diamantini@univpm.it (C. Diamantini); l.genga@tue.nl (L. Genga); c.gobbi@pm.univpm.it (C. Gobbi); a.mele@pm.univpm.it (A. Mele); d.potena@univpm.it (D. Potena)

ORCID 0000-0001-8143-7615 (C. Diamantini); 0000-0001-8746-8826 (L. Genga); 0009-0009-6589-5347 (C. Gobbi); 0009-0009-7852-0528 (A. Mele); 0000-0002-7067-5463 (D. Potena)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

with the original model [1]. These 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 behaviors not properly modeled; and so on.

Most state-of-the-art techniques neglect potential *correlations* among anomalous behaviors occurring at different stages of the process, repairing each of them independently. However, as discussed in [2], neglecting these correlations can result in including more behaviors than desired, compromising the quality of the final process model and allowing undesirable and potentially risky executions. For example, let us consider a loan application process where there exists a rule stating at which stage of the process the allocated employee should carry out a check to exclude fraud. In practice, it might be acceptable to postpone this check to just before sending the offer to the customer, due to various circumstances (e.g., holidays, or delays in getting the needed documents). Such violation will correspond to two anomalous behaviors in the event log, i.e., skipping the fraud check activity, and performing it in a not-allowed position. However, repairing these behaviors independently will lead to a process model in which the check can be skipped or executed arbitrarily multiple times, which is clearly undesirable.

In a previous work [3], we proposed an approach to address this issue by mining and repairing *high-level* anomalous behaviors (ABs), consisting of multiple low-level anomalous behaviors. Our experiments showed that considering high-level anomalies led to repaired models allowing for the behaviors of interest while maintaining precision and simplicity close to the original process model. In this demo paper, we introduce **ReLIGN**: a tool for model repair based on the approach proposed in [3]. The main functionalities of the proposed tool are detailed in the following section.

2. Innovation and functionality

The **ReLIGN** tool has been developed to support the repair approach introduced in [3], which consists of two main steps (see Figure 1). The approach takes as input a process model to be repaired, an event log tracking the process executions, and an AB to be used for the repairing step. Such behavior can either be extracted by means of ABs discovery techniques (e.g., [4]) or manually crafted by the user. First, a *Trace selection* step is performed. In this step, sequential traces are converted in the so-called *Instance Graphs* (IGs), which are Directed, Acyclic Graphs (DAGs) modeling the *control-flow* of a process execution considering both sequential and concurrent ordering relations among process activities. Our implementation leverages the approach proposed in [5] to generate the IG set. The AB is represented as a *Local Instance Graph* (LIG), that is, a subgraph of an IG capturing a specific behavior observed in the event log but not represented into the process model. Then, we select one trace to guide the repair, namely a trace whose corresponding IG contains the behavior represented by the LIG, therefore, the corresponding trace manifests the AB. This corresponds to searching for an *embedding* of the LIG in the IGs. In particular, we are interested in finding the *optimal embedding*, i.e., the trace whose corresponding IG contains the most similar subgraph to the AB provided in input, minimizing the number of additional anomalous events. We then proceed to the *AB integration* step, in order to perform the actual repair. This step aligns the AB to the process model, by

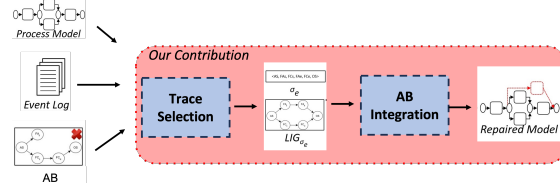


Figure 1: The proposed repair approach.

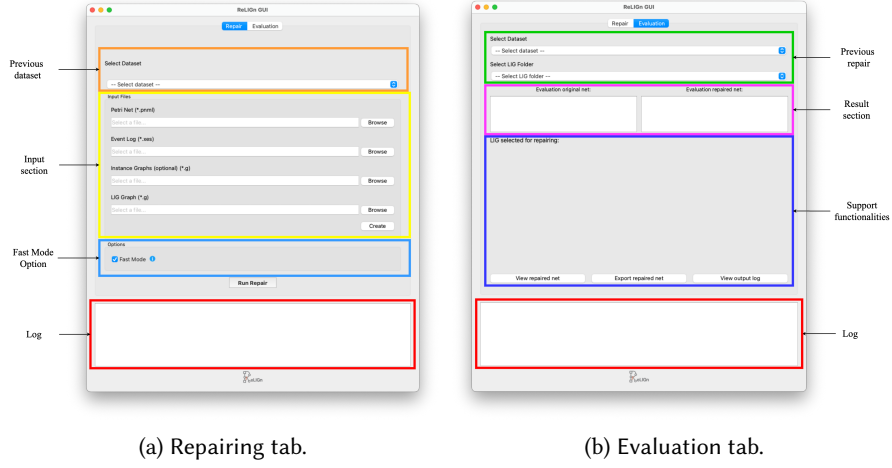


Figure 2: Repairing and evaluation tabs of the proposed tool.

(i) detecting the best location where the ABs should be placed, (ii) converting the graph in Petri Net notation and (iii) properly merging it with the original process model. We refer the interested readers to [3] for additional details.

ReLIGN delivers an end-to-end solution to perform process repair in a single run through automatic Instance Graph extraction and anomaly integration and offers an intuitive interface, ensuring complete transparency for the user. The interface (Figure 2) is formed by two tabs. the Repair tab lets the users choose the input parameters (yellow box in Figure 2a) configure the repair options (turquoise box in Figure 2a) and auto-populate the input fields with data from previous runs (orange box in Figure 2a). When the repair finishes, the Evaluation tab opens up automatically together with a visualization of the repaired net to show its results violet box in Figure 2b). The tab can be reopened at any moment to review previous runs (green box in Figure 2b). The tab also offers a Local Instance Graph viewer, an interactive Petri-net visualizer/exporter for the repaired model, and a detailed execution log (blue box in Figure 2b). The log box highlighted in red displays the state of the current execution in both tabs.

2.1. Repair tab

The Repair tab, using the defined input and configurations, generates the repaired process model (described as Petri net) according to the approach described in [3]. Once the repair

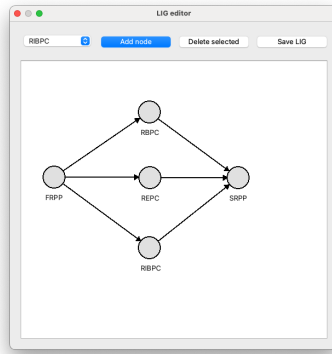


Figure 3: LIG editor functionality.

process is triggered, i.e., by clicking the Run Repair button, the repaired process model is displayed, and concurrently the Evaluation tab is shown in order to visualize the metrics.

2.1.1. Input selection

The entries on the yellow box of Figure 2 concern the selection of the dataset (event log in *.xes* format), the process model (Petri net in *.pnml* format), and, optionally, the dataset containing the IGs (in *.g* format). If IGs are not provided, the tool will execute the BIG algorithm at runtime to derive them according to the procedure described in [5]. The LIG can be provided (i) as input file (provided in *.g* format) or (ii) drawn in the LIG editor, as displayed in Figure 3.

2.1.2. Previous dataset

The drop-down menu highlighted in the orange box (Figure 2a) allows the user to select a dataset among those used in previous executions. This feature enables the system to automatically fill the input entries (yellow box, Figure 2a), preventing the user from having to manually insert the entries multiple times for a given dataset.

2.1.3. Fast mode option

To avoid potentially expensive and time-consuming computations, especially when working with large datasets, the user can enable the Fast Mode option (turquoise box, Figure 2a). When this option is selected, the repair procedure bypasses the identification of the optimal embedding, which implies the graph matching between the LIG and IGs of all traces in the event log. Instead, it stops at the first embedding found and selects that graph and its corresponding trace for the repair. However, this approach may result in selecting traces that exhibit additional ABs not represented by the original LIG. As a consequence, the final repaired model might be less accurate if compared to the one obtained by selecting the IG and the corresponding trace with the optimal embedding.

2.2. Evaluation tab

The Evaluation tab allows to visualize the results after repairing the process model. Figure 4 displays an example of output. As soon as the algorithm terminates, the repaired process model is displayed using two colors: the black is adopted for the original process model, while the red highlights the new elements that have been added by the repair algorithm.

2.2.1. Result section

The results section (violet box in Figure 2b) presents the metrics of both the repaired model and the original one. For each model, the following metrics were computed: fitness [6], which calculates the amount of behaviors occurring in the event log that is also allowed by the process model; precision [7], which evaluates how much of the behavior allowed by the model is actually observed in the event log; generalization, which assesses the model’s ability to represent not only the behavior seen in the log but also plausible future behavior; and model simplicity [8], which assesses the structural complexity of the model. All metrics were calculated using the standard implementations provided by the PM4Py suite [9].

2.2.2. Previous repair

The Previous repair section (green box in Figure 2b) allows to visualize the results of previous experiments by selecting both the dataset and the LIG of interest.

2.2.3. Support functionalities

The Support functionalities section (blue box in Figure 2b) provides a graphical visualization of the selected LIG used for the repair procedure. Additionally, it allows users to view and export the repaired process model in Petri net format (as a .pnml file), as well as to display a detailed log of the entire repair process.

3. Maturity and Availability

ReLIGn has been developed using Python 3.9.7¹. ReLIGn is available at <https://github.com/KDMG/ReLIGn-tool>. The tool has been tested on the same event logs used in the experimental section of [3], reporting the same results. A demonstration video-tutorial is available at <https://youtu.be/wafwsUPTees>.

Declaration on Generative AI

During the preparation of this work, the authors used GPT-4 in order to: Grammar and spelling check. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

¹<https://www.python.org/downloads/release/python-397/>

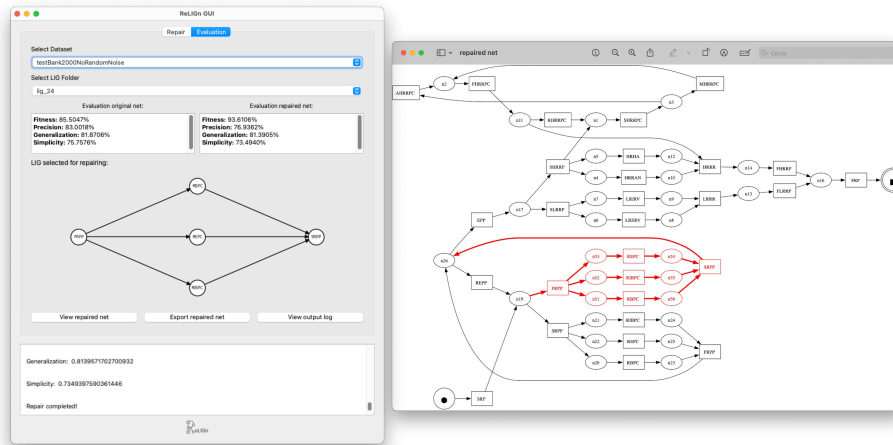


Figure 4: The output of an example repairing procedure.

References

- [1] D. Fahland, W. M. P. van der Aalst, Model repair - aligning process models to reality, *Information Systems* 47 (2014) 220–243.
- [2] A. Armas Cervantes, N. R. T. P. van Beest, M. La Rosa, M. Dumas, L. García-Bañuelos, Interactive and incremental business process model repair, in: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, Springer International Publishing, Cham, 2017, pp. 53–74.
- [3] L. Genga, F. Rossi, C. Diamantini, E. Storti, D. Potena, Model repair supported by frequent anomalous local instance graphs, *Information Systems* 122 (2024) 102349.
- [4] L. Genga, M. Alizadeh, D. Potena, C. Diamantini, N. Zannone, Discovering anomalous frequent patterns from partially ordered event logs, *Journal of Intelligent Information Systems* 51 (2018) 1–44.
- [5] C. Diamantini, L. Genga, D. Potena, W. M. P. van der Aalst, Building instance graphs for highly variable processes, *Expert Systems with Applications* 59 (2016) 101–118.
- [6] A. Adriansyah, B. F. van Dongen, W. M. 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.
- [7] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. Van Dongen, W. M. Van Der Aalst, Measuring precision of modeled behavior, *Information systems and e-Business Management* 13 (2015) 37–67.
- [8] F. R. Blum, Metrics in process discovery, University of Chile, Santiago, Chile, Report (2015) 1–21.
- [9] A. Berti, S. van Zelst, D. Schuster, Pm4py: A process mining library for python, *Software Impacts* 17 (2023) 100556.