Process-level Model Repair through Instance Graph Representation (Discussion paper)

Laura Genga[1], Claudia Diamantini[2,*], Emanuele Storti[2] and Domenico Potena[2]

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

Abstract
Existing model repair techniques propose changes that are based on event-level deviations observed in a log, like inserted or skipped events, often overlooking process precision at the advantage of fitness. The present short paper aims to briefly introduce the recent proposal of an alternative approach targeting higher-level structured anomalous behaviors. To do this, the approach exploits instance graph representation of anomalous behaviors, that can be derived from the event log and the original process model. The approach demonstrates that repaired models obtained in this way show higher precision and simplicity, with only small reduction of process fitness.

Keywords
Process Mining, Model Repair, Subgraph Mining

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

SEBD 2024: 32nd Symposium on Advanced Database Systems, June 23-26, 2024, Villasimius, Sardinia, Italy
*Corresponding author.
l.genga@tue.nl (L. Genga); c.diamantini@univpm.it (C. Diamantini); e.storti@univpm.it (E. Storti); d.potena@univpm.it (D. Potena)

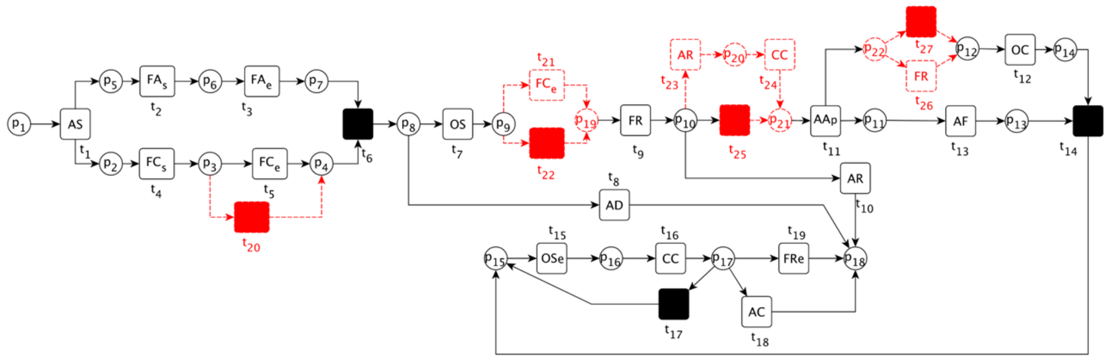


Figure 1: Loan Management Process model, original (solid black) and updated to include the discussed anomalous behavior (dashed red)

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 be addressed when repairing a process model is that of producing *high-quality* models. 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 [2]. However, as observed by [3], 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 present discussion paper presents an approach based on this idea, where anomalous behaviors are in fact sub-processes, represented in the form of instance graphs. Before entering into details of the approach, the following subsection introduces some basic concepts and discusses the issue by means of a motivating example.

1.1. Motivating example

Let us consider a simplified version of a loan management process derived from the BPI2012 challenge [4, 5]. 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.

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 also call them *high-level* anomalous behaviors, in contrast to *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. For instance, 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 execution traces which include respectively the first and the second practice.

Looking at single, low-level anomalies in isolation (i.e., in σ_1 FC_e before OS is missed, corresponding to a so-called deleted activity, FC_e unexpectedly appears before FR - inserted activity), state-of-the-art techniques [2] will return the model in Figure 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-occurence 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.A first step in this direction has been carried out by [3]. 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. The issue was recognized also in [2] where, besides the basic approach, an advanced version

[1]The model of the example has been repaired using the *ModelRepair* plugin implementation available at https://svn.win.tue.nl/repos/prom/Packages

that also caters for precision is 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. An example of repair of the loan management process based on the discovering of high-level anomalous behaviors is shown in Figure 2(c), as a preview of our contribution.

2. Methodology

Given the issues discussed in the motivating example, in a recent paper [6] we presented an approach aimed at defining a general methodology for automatically extracting structured anomalous behaviors from an event log, integrating them into the original model so to preserve precision and simplicity of the original model while improving fitness. The approach starts from the intuition that it is possible to model structured anomalies as (part of) process instances, that can be modeled as instance graphs thus exploiting frequent graph mining techniques for their discovery [7, 5]. The issue is then how to process these graphs, in order to incorporate them into the model correctly and effectively. The approach proposes 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, and (4) converts the graph in the given process model notation and properly merges it 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.

The proposed model repair methodology takes as input a process model M and an event log $\mathcal{L}$ recording process executions, and produces a repaired model M' on the basis of three steps:

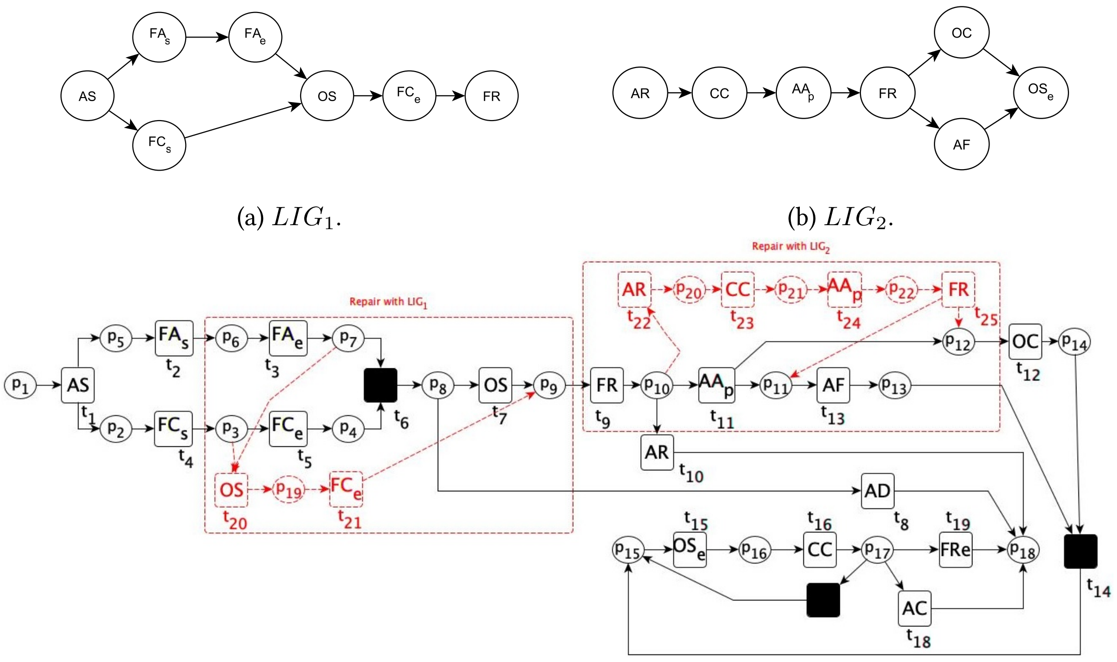
- *Anomaly discovery from event log:* in this step, Instance Graphs (IG) representations of log traces are first generated, to make concurrency among activities explicit. An IG is built for each trace in the event log by replaying the trace over the process model. For non-compliant traces, the procedure can lead to 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 IGs over-generalize the process behavior, allowing for much more behavior than observed in the log. To deal with the aforementioned issues, an *IG repairing* procedure is applied to IGs corresponding to anomalous traces, which transforms them in graphs capable of representing the anomalous traces without over-generalization [7]. Once repaired IGs are obtained, anomaly discovery based on frequent subgraph mining techniques is exploited in order to recognize subgraphs corresponding to frequent anomalous behaviors [5]. We will call them Local Instance Graphs (LIGs) in the following. The approach makes use of the SUBDUE algorithm [8];

- *Trace selection*: one trace σ is selected as to guide the repair from the set $\mathcal{L}_{LIG} \subseteq \mathcal{L}$ of traces containing the execution of the LIG. In particular, we select one of the traces with the *shortest extended embedding* of the LIG, which means the trace that contains the minimum number of additional anomalies beyond the one identified, so as 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 first aligns the LIG to the model and individuates the best location where it should be placed. Conformance checking is used to this end, to derive the best alignment η between the trace σ and M. Then, we derive σ_M, that is the projection of the alignment η on the model M. It contains all transitions in M which are in η, hence, σ_M conforms to the model. At this point, places of the process model M and events of LIG that will be connected to each other are identified. In particular, we scroll η until the first non-synchronous move occurring in the LIG is found. If a move-on-log is found, then the corresponding event is marked as $start_{LIG}$; otherwise (i.e., if a move-on-model is found), we scan η forward until a move corresponding to an event is found (i.e., either a synchronous move or a move-on-log), that is then marked as $start_{LIG}$. The transition of M corresponding to the last synchronous move before $start_{LIG}$ (i.e., $start_M$) and its position in σ_M are also computed. In order to find the connection points at the end of the LIG, the procedure is repeated considering reversed LIG and η. Hence, end_M and end_{LIG} are extracted. The places of the model to which to connect the LIG correspond to the markings reached after executing the transition $start_M$ and before executing end_M, respectively. Now, in order to keep the model as simple as possible, the LIG is modified by eliminating all nodes corresponding to transitions not to add to M. These are all events corresponding to synchronous moves in η occurring before/after $start_{LIG}/end_{LIG}$. Finally, the graph is converted in Petri Net notation and is properly merged with the original model.

Figures 2(a) and 2(b) show the anomalies described in Section 1.1 and identified in traces σ_1 and σ_2 respectively, while 2(c) reports the process model updated to include these anomalous behaviors.

We refer the interested reader to [6] for further details on the methodology.

We like to note here that the procedure has been conceived to incorporate the behavior of a single LIG instead of providing a global repair. 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, it makes sense to devise a simple procedure, which can be iterated to integrate other selected anomalies if



(a) LIG_1.

(b) LIG_2.

(c)

Figure 2: (a) LIG corresponding to the anomalous behavior identified in σ_1, (b) LIG corresponding to the anomalous behavior identified in σ_2, (c) the loan process model updated to include both LIG_1 and LIG_2.

necessary. In the absence of expert users and other domain-driven criteria, a notion of relevance can be, and has been, adopted as the criterion for LIG selection.

The methodology is based on the assumption that every activity in the log corresponds to a legal 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.

3. Discussion

A comprehensive set of experiments performed on synthetic and real-world event logs, that cannot be reported here for the lack of space (see [6] for details), shows that the repairing approach based on LIG in general outperforms the low-level repair in terms of precision and simplicity, at the cost of an often modest improvement of fitness. In details, the average gain in precision achieved by the proposed approach over the low-level one is 135.0% on synthetic datasets and 39.4% on real-world datasets. Regarding simplicity, the gain becomes 36.6% and 7.0% for synthetic and real-world logs, respectively. The results in terms of fitness are comparable with an average difference of 0.07 in favour of the low-level approach. The 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 [2], 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, which are characterised by high level of noise; in particular, as expected, with the increasing of the level of noise the drop in precision and simplicity increases as well, since models repaired by using the low-level approach involve much more noisy behaviors.

As a further observation, we note that in principle a low-level approach can be adopted to repair only specific behaviors. Similarly to what has been done in [9], 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, 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.

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 are more dependent on the characteristics of the LIGs. Indeed, if a LIG starts/ends with compliant behaviors, the proposed approach does not add these behaviors to the model, while the low-level repair adds 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 in our experiments is 54.98% on average, ranging from 29.9% to 83.6%. This leads to a percentage of duplicated transitions in the repaired model ranging from 6.1% to 13.2% (10.32% on average). 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, we use the Minimum Description Length criterium (implemented in the SUBDUE algorithm) to discover anomalous patterns, so as to ensure that the largest frequent common patterns are found first, contributing to the generalization ability. The first structures found should then be applied for repairing as a good practice to control overfitting. However, 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 event logs used in the experiments: 0.92 on average with a variance of 1.71E-03.

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 Description Length 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 [2] 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 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).

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 [2], 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 between these two alternative solutions.

References

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

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

[3] 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.

[4] A. Adriansyah, J. M. Buijs, Mining process performance from event logs: The BPI challenge 2012, BPM Center Report BPM-12-15, BPMcenter.org, 2012.

[5] 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.

[6] L. Genga, F. Rossi, C. Diamantini, E. Storti, D. Potena, Model repair supported by frequent anomalous local instance graphs, Information Systems 122 (2024) 102349. URL: https://www.sciencedirect.com/science/article/pii/S0306437924000073. doi:https://doi.org/10.1016/j.is.2024.102349.

[7] 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.

[8] I. Jonyer, D. J. Cook, L. B. Holder, Graph-based hierarchical conceptual clustering, Journal of Machine Learning Research 2 (2001) 19–43.

[9] 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.