Interactive and Incremental Business Process Model Repair

Abel Armas Cervantes[1], Nick R.T.P. van Beest[3], Marcello La Rosa[1], Marlon Dumas[2], and Luciano García-Bañuelos[2]

[1] Queensland University of Technology, Australia
[2] University of Tartu, Estonia
[3] Data61, CSIRO, Australia
{abel.armascervantes,m.larosa}@qut.edu.au
{luciano.garcia,marlon.dumas}@ut.ee
nick.vanbeest@data61.csiro.au

Abstract. It is common for the observed behavior of a business process to differ from the behavior captured in its corresponding model, as workers devise workarounds to handle special circumstances, which over time become part of the norm. Process model repair methods help modelers to realign their models with the observed behavior as recorded in an event log. Given a process model and an event log, these methods produce a new process model that more closely matches the log, while resembling the original model as close as possible. Existing repair methods identify points in the process where the log deviates from the model, and fix these deviations by adding behavior to the model locally. In their quest for automation, these methods often add too much behavior to the model, resulting in models that over-generalize the behavior in the log. This paper advocates for an interactive and incremental approach to process model repair, where differences between the model and the log are visually displayed to the user, and the user repairs each difference manually based on the provided visual guidance. An empirical evaluation shows that the proposed method leads to repaired models that avoid the over-generalization pitfall of state-of-the-art automated repair methods.

Keywords: Process model repair, conformance checking, visual analytics, process mining

1 Introduction

Modern information systems maintain detailed data about the execution of the business processes they support. These data can generally be extracted in the form of *event logs* consisting of sets of *traces*, i.e. sequences of *events* produced during the execution of a process case, where each event records the occurrence of a process activity.

Process mining is a family of methods for analyzing business processes based on event logs [24]. Among others, process mining methods allow analysts to compare the actual execution of a process against its expected execution captured in a process model. This model-to-log comparison operation is known as *conformance checking*. A conformance checking method takes as input a process model and an event log, and identifies a set of discrepancies between the behavior observed in the log and that allowed by the model. Once an analyst has identified relevant discrepancies between a process model and an event log via conformance checking, they may wish to modify the process model in order to better reflect reality. This operation is known as *process model repair*.

Process model repair methods take as input a process model, an event log and a set of discrepancies between the model and the log, and produce a model that resembles the original model as much as possible, but does not have the designated discrepancies.

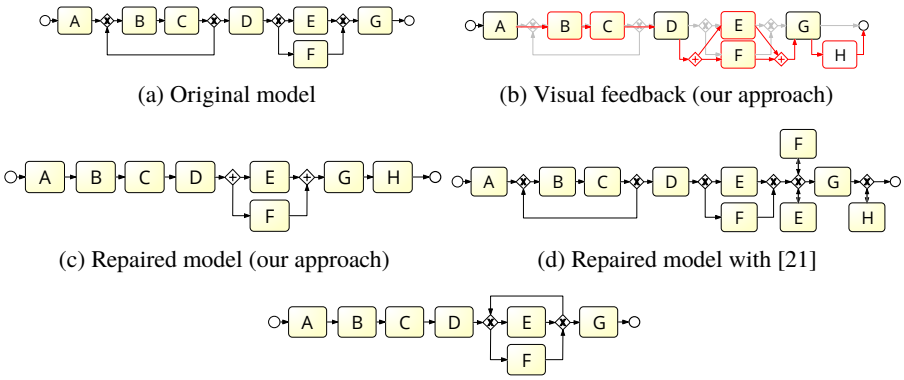
The quality of a process model repair method can be captured via three metrics: *structural similarity* (how much the produced model structurally resembles the original model), *fitness* (how much behavior observed in the event log is captured by the repaired process model), and *precision* (how much behavior is allowed by the repaired model but never observed in the log). A repaired model should be as structurally similar as possible to the original model, it should have a higher fitness than the original model (since the designated discrepancies are fixed) and it should not degrade precision (i.e. it should not add behavior that is not observed in the log).

Existing process model repair methods [10,21] identify points in the process where the log deviates from the model, and determine the model change operations required to reconcile such deviations. While the aim of [10] is to automatically generate repaired models with higher fitness w.r.t. the original model, [21] aims solely at finding the change operations to be performed. The latter method can associate different costs to the change operations and control the model change via a budget. Further, [21] shows that using only two change operations (insert and skip a task), which can be automatically applied over the model, it is possible to obtain a repaired model with higher fitness w.r.t. the original model. The authors in [21] acknowledge the importance of other metrics in addition to fitness, though the identification of techniques to improve on such metrics during repair is left to future work. In their quest for automation, these methods often add too much behavior to the original model, and thus the repaired model tends to grossly over-generalize the event log. In other words, these methods focus on maximizing fitness at the expense of precision.

This paper advocates for an *interactive* and *incremental* approach to process model repair for models in the BPMN language, where differences between the control-flow of the model and the log are visually displayed to the user and the user repairs each difference manually, at their discretion, based on visual guidance. In fact, some of these differences may underpin positive deviations, e.g. workarounds introduced to improve process performance, and as such, the user may wish to repair the model accordingly, so that these workarounds can become standard practices. On the other hand, differences that point to negative deviations, e.g. the violation of some compliance rule, should not be incorporated into the model. The paper purports that this approach allows users to strike a better tradeoff between the above metrics by giving them a more ample range of choices at each step of the repair process. This hypothesis is validated via an empirical evaluation on a battery of synthetic model-log pairs capturing recurrent change patterns as well as a real-life model-log pair.

To illustrate the benefits of our approach w.r.t. existing ones we consider the process model in Fig. 1a and the log $\{\langle A, B, C, D, E, F, G, H\rangle, \langle A, B, C, D, F, E, G, H\rangle\}$. Our approach provides the visual difference feedback to the user shown in Fig. 1b. Assuming this difference relates to a positive deviation, the user may repair the model in the way indicated in Fig. 1c. This model is similar to the original one, fixes the discrepancy shown in the visual feedback and does not add behavior w.r.t. the original model. Figures 1d and 1e are the repaired versions generated by the automatic repair methods in [21] and [10], respectively.[1] Even though the three models 1c - 1e can fully replay

[1] Both methods produce Petri nets but for simplicity we present the repaired models in BPMN.



(a) Original model

(b) Visual feedback (our approach)

(c) Repaired model (our approach)

(d) Repaired model with [21]

(e) Repaired model with [10]

Fig. 1: Examples of process model repairs using different approaches

the log (perfect fitness), the models 1d and 1e have lower precision. For instance, tasks B, C, E, F, H can be repeated any number of times in Fig. 1d.

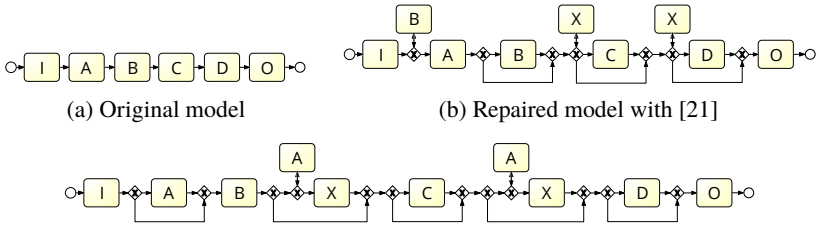
The paper is organized as follows. Section 2 discusses the limitations of existing process model repair methods. Section 3 introduces the conformance checking method presented in [12], which we use as a starting point. Next, Section 4 presents the proposed model repair approach, while Section 5 discusses the results of the empirical evaluation. Finally Section 6 summarizes the contribution and outlines future work directions.

2 Related work

Process model repair methods take as a starting point a set of discrepancies identified via a conformance checking method such as *trace alignment* [2,17]. This method computes a set of *optimal trace alignments* between each trace of a log and the closest corresponding trace of the model. An alignment is a pair of traces, which, in addition to symbols representing tasks, may also contain *silent moves*. A silent move represents a deviation between the trace of the log and the trace of the model. It may be a *move on log* (a task is observed in the log at a point where it is not allowed in the model) or conversely, a *move on model*. A trace alignment is optimal if it requires a minimum amount of moves.

In [10], the authors present a process model repair method based on alignments. This method starts by computing the optimal alignments between the log and the model, then identifies the non-conforming parts between them and, finally, adds i) loops, ii) subprocesses, and iii) skips of tasks. This approach guarantees a repaired model that fits the log perfectly. Another method, presented in [21], is also based on alignments. However, it only has two types of repair operations: skip a task and insert a new task loop. Unlike [10], this method seeks to maximize fitness while controlling the amount of changes by assigning a cost to each repair operation and setting a maximum budget.

The methods in [10] and [21] are based on change operations that add behavior to the model. While the former adds subprocesses, loops and skips, the latter adds skips and self-looping tasks. Although these changes have a positive impact on fitness, they negatively affect precision. Consider for example the model in Fig. 2a and log $\{\langle I, A, B, X, C, O\rangle, \langle I, A, B, X, D, O\rangle, \langle I, B, A, X, C, O\rangle, \langle I, B, A, X, D, O\rangle\}$. The re-



(a) Original model

(b) Repaired model with [21]

(c) Repaired model with [10]

Fig. 2: Examples of automatic process model repairs that over-fit (low precision)

paired models produced by [21] and [10] are shown in Figs. 2b and 2c, respectively. Even though both models perfectly fit the log, they allow additional behavior that is neither allowed in the original model nor observed in or implied by the log. For example, in Fig. 2b there are two tasks with label *X* that can be repeated before and after *C*, while in Fig. 2c, the same applies to task *A*. Furthermore, in both repaired models, tasks *C* and *D* can co-occur, even though this never happens in the log.

The work in [7] proposes a genetic algorithm that given a reference model and a log, discovers a new model that is similar to the reference model and more closely fits the log. This method optimizes the result along five dimensions: fitness, precision, simplicity, generalization and structural similarity to the reference model. The method generates candidate models (represented as process trees), which are evolved until one of them is found to be optimal w.r.t. a given threshold on the allowed changes. Unlike process model repair methods, this method does not update the original model but discovers a new (possibly very different) one.

The authors of [22] consider the problem of model repair in a context where the log is not necessarily reliable (i.e. there is missing or incorrect data). Hence the analyst needs to indicate to what extent they trust the model and to what extent they trust the log. If the analyst trusts the model but not the log, the log is fully re-generated so that it matches the original model. If the analyst trusts the log but not the model, the model is re-discovered from the event log using an automated process model discovery method, creating a model that can potentially be very distant from the original one. If the user partially trusts the model and the log, a new (repaired) model-log pair is generated, such that the repaired model and log match, and their respective behavior is within a certain distance of the original model and log.

Finally, process model repair has also been approached in the context of unsoundness. For example, [11] uses a multi-objective optimization technique to automatically turn an unsound Petri net into a corresponding sound model in a minimal number of change operations, in an attempt to keep a high structural similarity to the original model. In this case, however, the input to the technique is only an (unsound) model.

3 Behavioral alignment

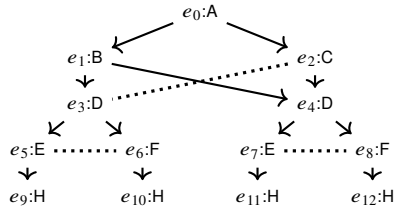
Behavioral alignment [12] is a conformance checking method that identifies events or behavioral relations between tasks occurrences that are observed at a given point in the log but not allowed in the corresponding state of the model, and vice-versa. This method operates in four steps. First, it compresses the event log into a graph of behavioral relations between task occurrences known as an *event structure* [19].

Second, it expands the process model into another event structure. Third, it computes a (partially) synchronized product between these two event structures. Finally, it extracts a set of difference statements from the product.

A Prime Event Structure (PES) [19] is a graph of events representing occurrences of tasks. Events are linked via three relations: i) *Causality* ($e < e'$) indicates that event e is a prerequisite for e'; ii) *Conflict* ($e \# e'$) implies that e precludes the execution of e'; and iii) *Concurrency* ($e \parallel e'$) indicates that e and e' co-occur in any order. A PES represents the computations of a system by means of *configurations*, sets of events that can occur together. Fig. 3b shows the PES extracted from the log shown in Fig. 3a. A label e:A represents an event e signifying an occurrence of task A. For example, e_9:H, e_{10}:H, e_{11}:H and e_{12}:H are events representing different occurrences of task H. A directed black arc between events represents causality, while a dotted edge represents conflict. Concurrency is denoted by the lack of any (direct or transitive) link between two events, e.g. events e_1:B and e_2:C are concurrent. The PES of a log represents every trace in the log as a configuration. E.g., the first trace in Fig. 3a is represented by the configuration $\{e_0$:A, e_1:B, e_2:C, e_4:D, e_7:E, e_{11}:H$\}$.

Trace
A B C D E H
A B C D F H
A C B D E H
A C B D F H
A B D E H
A B D F H

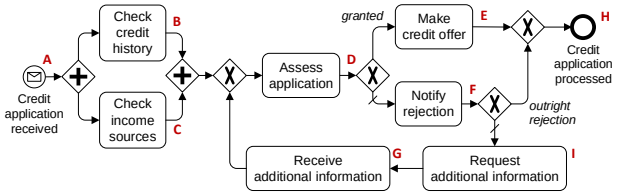
(a) Event log



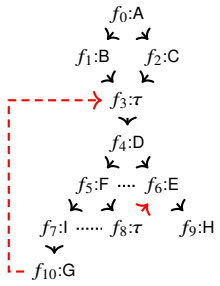
(b) Induced PES

Fig. 3: Example of an event log and its corresponding PES

Any BPMN process model can be expanded into a PES using a technique known as unfolding. In this alternative representation of the behavior of the model, each event is associated to a single task, thus it is always possible to determine which task originated a given event. Consider the loan application process model shown in Fig. 4a, which is unfolded into a corresponding PES in Fig. 4b. For conciseness, the PES prefix uses short labels A, B, ..., I (shown next to each task in the BPMN model) instead of the full task labels. The PES of a model can also contain *cutoff-corresponding event relations*,



(a)



(b)

Fig. 4: Loan application as BPMN model (a) and PES prefix (b)

graphically represented as directed red dotted arrows (see Fig. 4b), to denote "jumps" in the continuation of the process execution (e.g., in the case of looping behavior).

The event structure mirrors the BPMN model: the first event is $f_0{:}A$ and is followed by concurrent events $f_1{:}B$ and $f_2{:}C$. A silent event ($f_3{:}\tau$) captures their synchronization. Subsequently, $f_4{:}D$ is followed by two events in conflict ($f_5{:}F$ and $f_6{:}E$). Event $f_6{:}E$ can only be followed by $f_9{:}H$, while $f_5{:}F$ can be followed either by event $f_7{:}I$ (itself followed by $f_{10}{:}G$) or by a silent event $f_8{:}\tau$ which is in conflict with $f_7{:}I$. The cutoff-corresponding relation ($f_{10}{:}G$, $f_3{:}\tau$) tells us that the process can jump back to the point just before D is executed, while ($f_8{:}\tau$,$f_6{:}E$) captures the jump from F to H.

The event structure of the log and that of the model are compared by computing a *partially synchronized product* (PSP) [4,12]. A PSP is a state machine where each state is a triplet $\langle C^l, C^r, \xi\rangle$, where C^l is a configuration of the PES of the log, C^r is a configuration of the PES of the model, and ξ is a partial mapping between them. The construction of the PSP starts with the empty configurations. At each step, a pair of events from each PES is matched (added to ξ) if and only if their labels are the same and their causal relations with the other matched events coincide. When an event cannot be matched, it is "hidden" to allow the comparison to proceed. A "hide" edge captures behavior observed in the log but not allowed in the model ($lhide$), or behavior allowed in the model but not observed in the log ($rhide$). The method relies on an A^* heuristic to build a PSP that finds, for each configuration in the log, the most similar configuration in the model (a.k.a. *optimal*), i.e., the one with the minimum number of hides.

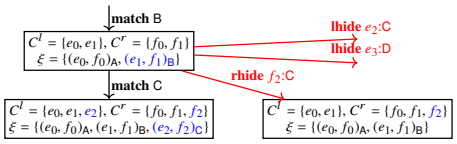


Fig. 5: Fragment of the PSP for PES from Figs. 3b and 4b

Figure 5 presents an excerpt of the PSP of the events structures in Figs. 3b and 4b. The topmost box is the state where configurations $C^l = \{e_0, e_1\}$ (log PES) and $C^r = \{f_0, f_1\}$ (model PES) have been processed, resulting in the mapping $\{(e_0, f_0)_\mathsf{A}, (e_1, f_1)_\mathsf{B}\}$. Given the above state, the events $\{e_2{:}\mathsf{C}, e_3{:}\mathsf{D}\}$ from log PES would be enabled, and so is $f_2{:}\mathsf{C}$ from the other PES. Under such conditions, four moves are possible in the PSP: (i) the matching of events e_2 and f_2, both carrying the label C, (ii) the (left) hiding of $f_2{:}\mathsf{C}$, and (iii) the (right) hiding of $e_2{:}\mathsf{C}$ and $e_3{:}\mathsf{D}$. Fig. 5 presents only the states reached after operations "match C" and "rhide $f_2{:}\mathsf{C}$". However, the full PSP will contain optimal matchings for all runs in the PES of the event log.

The dissimilarities between a model and a log can be of two types, mismatching behavior and behavior only observed in the model (not observed in the log); while the former is captured by means of hide operations in the PSP, the latter is the model behavior not observed in the log. The mismatching behavior patterns defined in [12] are shown in Fig. 6, while their verbalizations are displayed in Table 1. In the verbalizations, the capital letters are placeholders for the activities involved in the differences. Note that UnobsAcyclicInter and UnobsCyclicInter in Table 1 refer to the behavior solely contained in the model.

Table 1: Verbalization of mismatch patterns

Pattern	Statement
TaskReloc	*In the log, B occurs after* $[A]$ *instead of* $[A, C, D]$
ConcConf	*In the model, after* $[A]$*, B and C are concurrent, while in the log they are mutually exclusive*
CausConc	*In the model, after* $[A]$*, B occurs before C, while in the log they are concurrent*
CausConf	*In the model, after* $[A]$*, B occurs before C, while in the log they are mutually exclusive*
TaskSub	*In the log, after* $[A]$*, B is substituted by X*
TaskAbs/Ins	*In the log, C occurs after* $[A, B]$ *instead of* $[A, C]$
UnmRepetition	*In the log, A is repeated after* $[B]$
TaskSkip	*In the log, after* $[A]$*, B is optional*
UnobsAcyclicInter	*In the log, tasks* $[A, B, \ldots]$ *do(es) not occur after tasks* $[D, E, F]$
UnobsCyclicInter	*In the log, the cycle involving tasks* $[A, B, \ldots]$ *does not occur after tasks* $[D, E, F]$

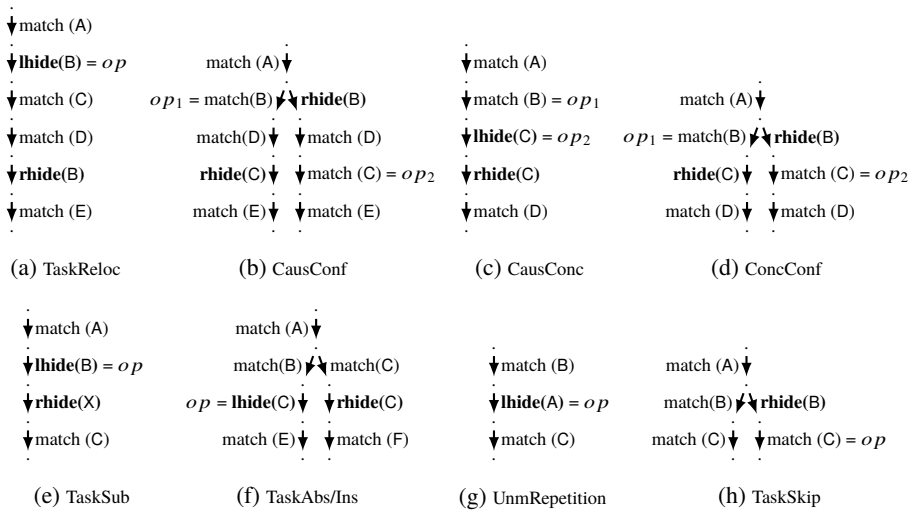


Fig. 6: Mismatch patterns in the PSP

Roughly speaking, given a PSP both the mismatching behavior patterns and the additional behavior in the model are identified, verbalized and reported. For instance, the fragment of the PSP shown in Fig. 5 captures the pattern in Fig. 8c that is verbalized as *"In the event log, task* C *can be skipped, while in the model it cannot"*.

4 Extending conformance checking to process model repair

In this section, we first describe two improvements over the behavioral alignment method introduced above: four of the existing patterns are redefined to consider sequences of tasks (also referred to as *intervals*) instead of individual tasks, and an order for the detection of differences is established in a way that more specific patterns are detected before more general ones. Next, we describe a method for interactive process model repair based on the visualization of the differences detected by the revised behavioral alignment method.

4.1 Extension, order and impact of differences

The patterns in Table 1 define combinations of hide and match operations to describe predefined templates expressing behavioral differences. Those patterns capture differences

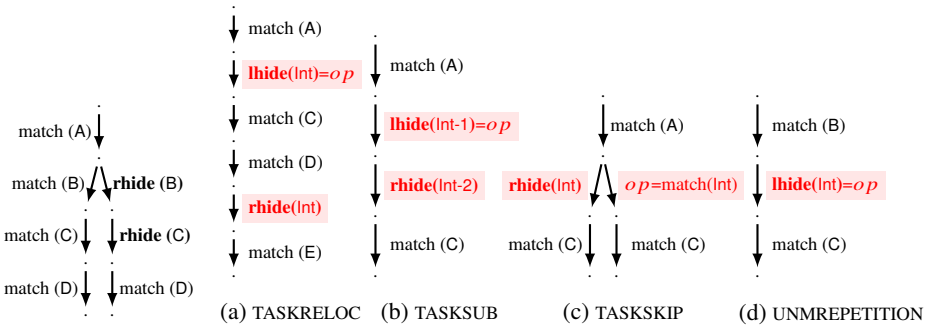


Fig. 7: Interval skip

Fig. 8: Patterns extended to consider intervals

involving single tasks (TaskSkip, TaskSub, UnmRepetition, TaskReloc and TaskAbs), pairs of tasks (CausConc, ConcConf and CausConf) or even sequences of tasks (UnobsAcyclicInter and UnobsCyclicInter). Patterns involving sequences of tasks can offer condensed dissimilarities, which otherwise should be spelled out one by one. Thus, as a first contribution, we redefined four patterns, those displayed in Fig. 8, to consider sequences of tasks instead of single tasks. In addition, to obtain a more condensed feedback, the new definitions fix some of the issues of the original definitions. For example, even though the PSP in Fig. 7 shows the case when tasks B and C are optional (i.e., they occur in both the model and the log, or only in the model), the original definition considers only the optionality of a single task.

An interval (Int) is a sequence of tasks that can occur one after the other either in the model or in the log. For instance, the occurrences of tasks B and C in Fig. 7 represent an interval, because they occur consecutively in the model after A. For the sake of brevity, the $rhide$, $lhide$ or $match$ of an interval Int is denoted as $rhide(Int)$, $lhide(Int)$ or $match(Int)$, respectively. Thus, given an interval $Int = [B, C]$, $rhide(Int)$ denotes $rhide(B)$ followed by $rhide(C)$, and analogously for $lhide$ and $match$. The modified patterns to consider intervals instead of single tasks are depicted in Figure 8. Observe that in the case of Fig. 8b, Int-1 and Int-2 are two different intervals.

The detection of differences consists of traversing the PSP and once a pattern is found, all the involved hide operations are discarded. As such, a hide operation cannot be reported as part of two differences. However, a single hide operation can be explained by various patterns. For example, given the PSP in Fig. 7, it is possible to identify two TaskAbs patterns (i.e., *"In the model, B occurs after [A] and before [D]"* and *"In the model, C occurs after [A] and before [D]"*) or a single TaskSkip pattern (i.e., *"In the model, after [A], the interval [B, C] is optional"*). The latter offers a deeper insight into the difference, in a more compact manner, and is thus preferred. Therefore, as a second contribution, we define a specific order on the detection of the differences, such that those involving intervals are identified first, then differences involving pairs of tasks are identified, and finally those involving only single tasks. The order for each of the patterns is shown in the first column of Tables 2-4.

The proposed repair approach starts from the premise that not all existing differences shall be reconciled but only those pointing to positive deviations, and among them, some may have higher priority than others. For example, differences involving critical tasks, or differences that refer to a particular type of mismatch pattern (e.g., TaskAbs), or affecting a larger number of traces in the log, may be given higher priority. In this

regard, as a third contribution we propose a notion of *impact* for each mismatch pattern. We define this notion based on the frequency of the events in the log, such that differences with higher impact shall be reconciled first.[2] The construction of the PES enriched with information about the frequency of the events, can be found in [23]. Consider the patterns in Fig. 6 and a log with X amount of traces. The impact of a given pattern is defined as Y/X, where Y is i) the frequency of the event (or the minimum frequency of the events in an interval) involved in the operation *op* for the patterns **TaskReloc**, **TaskSub**, **TaskAbs/Ins**, **UnmRepetition** and **TaskSkip**; ii) the minimum frequency of the event in op_1 and in op_2 for the patterns **CausConf** and **CausConc**; and iii) the frequency of the event in op_1 plus the frequency of the event in op_2 for the pattern **ConcConf**. Intuitively, this notion represents the proportion of traces involving the events in a given difference. This simple impact measure can be replaced with a more sophisticated notion, e.g. one that depends on factors that are exogenous to the log such as the cost of rectifying the model according to a given difference.

4.2 Visualization of differences

The cornerstone of our interactive and incremental model repair approach is the visualization of the differences. It exploits the fact that diagrams are powerful tools for presenting information in a more concise and precise manner than text [18]. Thus, the visualization of differences can be easier to understand than the textual description generated by the conformance checker. Intuitively, every mismatch pattern is translated into a graphical representation, which can be overlaid on the model, and suggests the change to be done for reconciling a given difference. This alternative representation uses standard BPMN notation, so no new symbols are added, and uses a color code to represent the suggested changes. Variations in color are easily distinguishable, more than changes in shapes [18], and can help coping with potential model complexity. This idea incarnates the principle of *graphical highlighting*, which have been shown in [15] to lead to more understandable process models.

Some techniques that have approached the problem of representing differences between graphs and/or models can be broadly categorized into two groups: those that use color-coding of differences in a merged graph for the visualization of differences (see e.g. [14,6,13,20]), and those that overlay the two compared models, such that both models are visible in the same picture [3]. In the visualization proposed in this paper, however, the differences are not directly represented as such to the user. Rather, the required *change* is shown, indicating what needs to be changed in order to repair the model such that it matches the behavior observed in the log.

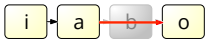
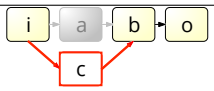
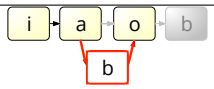
The color code used for the representation of the differences is as follows. An element in the BPMN model – task, sequence flow (i.e. arc) or gateway – involved in a difference can either be grey (element to be removed) or red with thicker lines (element to be inserted or task affected by the difference); whereas the elements that are not involved in the difference are left unchanged. The proposed visualizations for all mismatch patterns are displayed in Tables 2 - 4. For instance, consider the fourth pattern in Table 2, where task b has to be relocated after o. As a result, task b in the model is grayed out (along with its incoming and outgoing arcs), while a new task b is inserted after o. The background of the new task is colored white, so that it is easier to distinguish new tasks from those already present in the model (those with a yellow background).

[2] The same rationale of reconciling changes with higher impact first is proposed in [21].

The differences are classified with respect to two criteria, *scope* and *type* of change. The scope can be local or cross-context. Local changes occur in a single part of the model, e.g., **TaskAbs** denotes the case when a task does not occur in the log, and thus it has to be removed from the model; whereas, cross-context changes involve two different parts in the model, e.g., **TaskReloc** represents the case when an interval of tasks has to be relocated in the model. The second criterion is the type of change: tasks modification, sequence flows modification, or gateways modification. In our context, a modification implies either the removal, or the removal and insertion of elements.

A local change can be further subdivided into two classes: *interval* and *binary*. A local-interval change can be formally defined as a triplet $\langle I_1, I_2, C\rangle$, such that I_1 is an interval of tasks in the model, I_2 is an interval of tasks to be inserted and C is a configuration. A local-binary change is a triplet $\langle e_1, e_2, C\rangle$, where e_1 and e_2 are tasks in the model, and C is a configuration. Finally, a *cross context* change is defined as $\langle I, C_1, C_2\rangle$, where I is an interval, and C_1 and C_2 are configurations. C_1 is the source of the difference and C_2 is the target of the modification. We refer to the start (resp. end) of a difference as the element in the model that precedes (resp. follows) the tasks in I_1, I_2, $\{e_1, e_2\}$ and I, depending on the scope of the change. In order to build the visualization of a difference, we take as input the triplets generated by the conformance checker, and obtain the elements in the model involved in such differences. Subsequently, we generate a triplet $\langle Y, H, A\rangle$, such that Y contains the elements to be grayed out, H contains the elements to be highlighted and A contains the elements to be added. Each set of patterns, grouped by the type of change in the model, is presented below.

Table 2: Tasks modification patterns

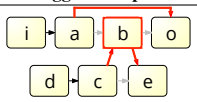
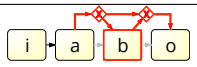
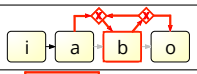
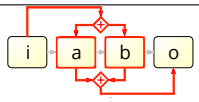
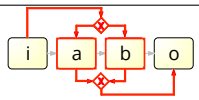
Order	Type	Input	Suggested repair	Sentence
16	TaskAbs	$I_1 = \{b\}$ $I_2 = \emptyset$ $C = \{i, a\}$		In the model, Task *b* occurs after [*i, a*] and before [*o*]
6	UnobsAcyclic	$I_1 = \{a, b\}$ $I_2 = \emptyset$ $C = \{i\}$		In the log, the interval [*a, b*] does not occur after [*i*]
8	TaskSub	$I_1 = \{a\}$ $I_2 = \{c\}$ $C = \{i\}$		In the log, after *i*, the interval [*a*] is substituted by Task *c*
1	TaskReloc	$I = \{b\}$ $C_1 = \{i, a\}$ $C_2 = \{i, a, b, o\}$		In the log, the interval [*b*] occurs after [*i, a, o*] instead of [*i, a*]
2	TaskReloc	$I = \{b\}$ $C_1 = \{i, a, o\}$ $C_2 = \{i, a\}$		In the model, the interval [*b*] occurs after [*i, a, o*] instead of [*i, a*]

Tasks modification. This set of differences covers three cases: tasks need to be removed from the model, tasks need to be relocated and tasks need to be substituted. Table 2 presents the patterns in this category and shows an example of both their visualization and their verbalization. The graphical representation of these differences consists in graying out the tasks (and their arcs) in the interval I_1 for the local-interval changes, and the tasks in the interval I for the cross-context changes. New arcs, and tasks in the case of task substitution, are inserted to connect the start with the end of each difference (i.e., the tasks around the grayed out elements). Finally, for the last two cross-context changes, new tasks and arcs are inserted representing the relocation of the intervals to

the target of the modification. For instance, the fourth pattern – **TaskReloc** in Table 2 suggests to relocate task b after $\{i, a, o\}$. Thus, task b and its incoming/outgoing arcs are grayed out, a new arc is inserted to connect a with o, and a new task b is added after o.

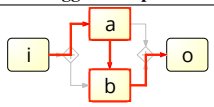
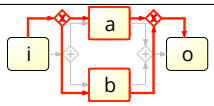
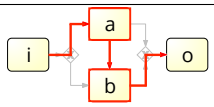
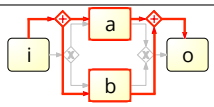
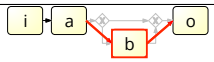
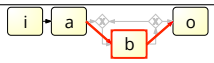
Sequence flows modification. The differences in this category cover the cases when existing arcs need to be removed, and new gateways and/or arcs need to be inserted. Thus, no task needs to be grayed out or highlighted. The patterns in this category are presented in Table 3. The arcs to be grayed out are the incoming and outgoing arcs of the tasks in I for cross-context changes, in I_1 for local-interval changes, and in $\{e_1, e_2\}$ for local-binary changes. Finally, depending on the pattern, new gateways have to be inserted with corresponding arcs for connecting the elements involved by the difference. For instance, the second pattern (**TaskSkip**) in Table 3 suggests that task b should be optional. The incoming and outgoing arcs of b are grayed out, and new XOR gateways are added to allow the skip of b after a.

Table 3: Sequence flows modification patterns

Order	Type	Input	Suggested repair	Sentence
15	TaskAbs	$I = \{b\}$ $C_1 = \{i, a\}$ $C_2 = \{d, c\}$		In the log, Task b occurs after [d, c] and before [e]
3	TaskSkip	$I_1 = \{b\}$ $I_2 = \emptyset$ $C = \{i, a\}$		In the log, after a, the interval [b] is optional
7	UnmRepetition	$I_1 = \{b\}$ $I_2 = \emptyset$ $C = \{i, a\}$		In the log, the interval [b] is repeated after a
9	CausConc	$e_1 = a$ $e_2 = b$ $C = \{i\}$		In the model, after i, Task a occurs before Task b, while in the log they are concurrent
14	CausConf	$e_1 = a$ $e_2 = b$ $C = \{i\}$		In the model, after i, Task a occurs before Task b, while in the log they are mutually exclusivethe log they are concurrent

Gateways modification. The last set of differences are changes that affect the gateways present in the model, i.e. a gateway needs to be deleted or replaced by another gateway (e.g. an XOR gateway is replaced by an AND gateway). The patterns in this category are shown in Table 4. The elements to be highlighted are the tasks in the interval I and I_1 in the case of **TaskSkip** and **UnobsCyclic**, respectively, and tasks e_1 and e_2 for the rest of the differences. The elements grayed out are the relevant gateways (AND gateways for the first two patterns and XOR gateways for the last four), and their outgoing and incoming arcs. Finally, depending on the pattern, new gateways and arcs are inserted to connect the elements involved in the pattern. For example, the first and third patterns in Table 4 suggest to remove the gateways and to define a causality order between a and b. In these two patterns, a (the task occurring first) is connected to i, and b (occurring last) is connected to o. In the case of the second and the fourth patterns in Table 4, the existing gateways have to be substituted by another type, thus changing the parallel behavior between tasks a and b to exclusive, or vice versa. The substitution is as follows: existing gateways and their incoming and outgoing arcs are grayed out, and new gateways and arcs are inserted. Specifically, in both **ConcConf** patterns, the fork gateway is connected

Table 4: Gateways modification patterns

Order	Type	Input	Suggested repair	Sentence
10	CausConc	$e_1 = a$ $e_2 = b$ $C = \{i\}$		In the log, after i, Task a occurs before Task b, while in the model they are concurrent
12	ConcConf	$e_1 = a$ $e_2 = b$ $C = \{i\}$		In the model, after i, Task a and Task b are concurrent, while in the log they are mutually exclusive
13	CausConf	$e_1 = a$ $e_2 = b$ $C = \{i\}$		In the log, after i, Task a occurs before task Task b, while in the model they are mutually exclusive
11	ConcConf	$e_1 = a$ $e_2 = b$ $C = \{i\}$		In the log, after i, Task a and Task b are concurrent, while in the model they are mutually exclusive
4	TaskSkip	$I = \{b\}$ $C_1 = \{i, a\}$ $C_2 = \{i, a, o\}$		In the model, after i, the interval $[b]$ is optional
5	UnobsCyclic	$I_1 = \{b\}$ $I_2 = \emptyset$ $C = \{ı, a\}$		In the log, the cycle involving $[b]$ does not occur after $[i]$

to i, a and b, emulating the connections of the fork gateway to substitute, while the join gateway is connected to a, b and o.

5 Evaluation

We implemented our approach as part of the OSGi plugin *Compare* [5] for the Apromore online process analytics platform.[3] This plugin takes as input a BPMN process model and a log in MXML or XES format. Its output is a set of textual differences according to the mismatch patterns described in this paper. When selected, a difference is also represented graphically on top of the input process model. The user can apply a difference at a time to repair the model accordingly. At each application, the differences between the model and the log are recomputed. Once a difference has been selected, it can be automatically applied by the tool. Users can apply differences until the model and the log capture the same behavior, or until desired.[4] An example of the graphical representation of a difference, i.e., **TaskSkip** pattern, over a model is depicted in Figure 9.

Using this tool, we conducted a two-pronged evaluation to compare our approach to the two existing baseline approaches in [21] (hereafter called Base1) and [10] (Base2). First, we applied each approach on a battery of synthetic event logs generated from a real-like model of a loan origination process, to assess how each of them performs in identifying and repairing elementary changes. Next, we applied each approach on a real-life model-log pair for a road traffic fines management process.

We compared the quality of the repaired models produced by the three approaches in terms of their fitness, precision and F-Score (the harmonic mean of fitness and precision)

[3] Available at `http://www.apromore.org`

[4] A screencast of the tool can be found at `https://youtu.be/3d00pORc9X8`

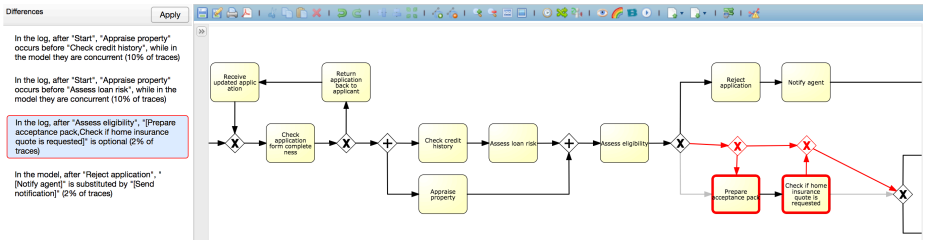


Fig. 9: Example of the visualization of a difference in *Compare*

w.r.t. the event log, as well as in terms of their structural similarity w.r.t. the original model. We used fitness and precision based on trace alignment [2,1], as these metrics can be computed reasonably quickly, and because the two baseline methods were designed to optimize the fitness measure based on trace alignments. Trace alignment-based fitness [2] measures the degree to which every trace in the log can be aligned with a trace produced by the model, while trace alignment-based precision [1] measures how often the model escapes these aligned traces by adding extra behavior not recorded in the log. We computed model similarity as one minus the graph-edit distance between the two models. The graph-edit distance measures the number of node and edge insertions, removals and substitutions to transform one graph into the other. We used the measure in [8] (with a greedy matching strategy), as it has been shown to provide a good compromise between matching accuracy and performance.

5.1 Experiment with synthetic datasets

In the first experiment, we used a battery of 17 synthetic model-log pairs. Starting from a textbook example of a process for assessing loan applications [9] (see Fig. 10), we generated 17 altered versions of this base model by applying different change operations. Next, we used the BIMP simulator[5] to generate an event log from each altered process model. By pairing the base model with these logs, we obtained 17 model-log pairs.

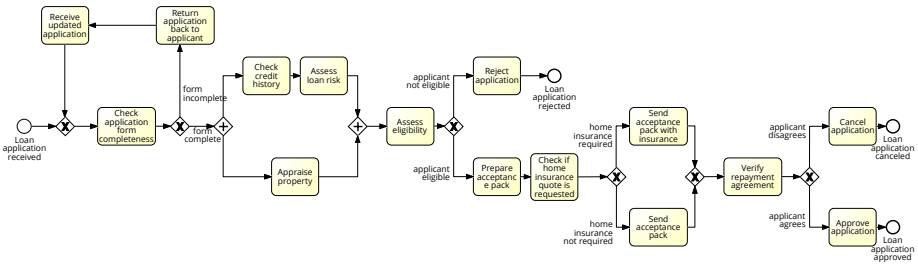


Fig. 10: BPMN model of a loan origination process (source: [9])

To avoid bias towards any of the evaluated approaches, we selected the change operations to apply from an independent taxonomy of *simple change patterns* [25], which constitute solutions for realizing commonly occurring control-flow changes in information systems. As such, this taxonomy of changes is different from the mismatch

[5] `http://bimp.cs.ut.ee`

patterns presented in this paper, which are based on the difference as observed in the PSP. However, each pattern in one taxonomy can be expressed by one or more patterns in the other taxonomy.

The simple change patterns from [25], summarized in Table 5, capture elementary ways of modifying a process model, such as adding/removing a fragment, putting a fragment in a loop, swapping two fragments, or parallelizing two sequential fragments. Non-applicable patterns such as changing branching frequency or inlining a subprocess were excluded, resulting in eleven change patterns. These patterns can be grouped into three categories based on their type: Insertion ("I"), Resequentialization ("R") and Optionalization ("O"). From these categories, following the same method as in [16,23], we constructed six *composite change patterns* by subsequently nesting simple change patterns from each category within each other: "IOR", "IRO", "OIR", "ORI", "RIO", and "ROI". For example, the composite pattern "IRO" can be obtained by adding a fragment ("I"), putting it in parallel with an existing fragment ("R"), and skipping the latter ("O"). As a result, we obtained a total of 17 change patterns.

Table 5: Simple control-flow change patterns from [25]

Simple pattern	**Explanation**	**Category**
Add/Remove	Add/remove fragment	I
Cond./Seq.	Make two fragments conditional/sequential	R
Conc./Seq.	Make two fragments concurrent/sequential	R
Loop	Make fragment loopable/non-loopable	O
Skip	Make fragment skippable/non-skippable	O
Cond. move	Move fragment into/out of conditional branch	I
Conc. move	Move fragment into/out of concurrent branch	I
Synchronize	Synchronize two parallel fragments	R
Duplicate	Duplicate fragment	I
Replace	Substitute fragment	I
Swap	Swap two fragments	I

In order to use our approach without user input, we automatically selected the difference retrieved from our tool that has the highest impact in terms of involved log traces, applied that to the original model to obtain a repaired model, recomputed the differences and picked again the most impactful difference, until no more differences existed or five differences had been selected. This *mechanized* version of our approach only removes the interaction with the user but preserves its incremental nature. However, given the limit to maximum five differences, there is no guarantee that all the discrepancies between model and log would be repaired.

Table 6 reports the results of the first experiment, where for each bidirectional pattern (e.g. Add/Remove), we applied the pattern in both directions and reported the average measurements. Our approach always achieves the highest structural similarity w.r.t. the original model (on average 0.92, with values ranging from 0.86 to 0.97). This is substantially higher than the similarity obtained by Base1 (avg=0.72, min=0.60, max=0.88) and by Base2 (avg=0.79, min=0.71, max=0.88). Despite the higher similarity, the models repaired by our approach have the highest F-Score in all but one case. In fact, while both baselines aim to maximize fitness, obtaining a perfect fitness in most cases, our approach keeps the fitness high while improving precision, often substantially (avg=0.97 against 0.68 for Base1 and 0.94 for Base2), hence striking a better balance between the two accuracy measures. The only exception is the synchronization pattern,

where our F-Score is 0.95 against 0.97 for the two baselines. The repair introduced in this case was more specific than the behavioral difference reported by our approach, resulting in a lower fitness compared to the two baselines (0.94 instead of 1.00), despite having a slightly higher precision (0.97 instead of 0.95).

Table 6: Evaluation results on the synthetic datasets

		Add/Remove	Cond./Seq.	Conc./Seq.	Loop	Skip	Cond. move	Conc. move	Synchronize	Duplicate	Replace	Swap	IOR	IRO	OIR	ORI	RIO	ROI
Base1 [21]	**Similarity**	0.74	0.68	0.88	0.57	0.82	0.72	0.70	0.88	0.67	0.73	0.63	0.60	0.70	0.62	0.82	0.80	0.72
	Fitness	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Precision	0.68	0.94	0.92	0.36	0.87	0.70	0.72	0.95	0.52	0.64	0.55	0.59	0.45	0.46	0.92	0.84	0.50
	F-**Score**	0.81	0.97	**0.96**	0.53	0.93	0.82	0.84	**0.97**	0.69	0.78	0.71	0.74	0.62	0.63	0.96	0.91	0.67
Base2 [10]	**Similarity**	0.79	0.79	0.88	0.78	0.77	0.75	0.74	0.88	0.82	0.79	0.71	0.82	0.88	0.71	0.79	0.83	0.74
	Fitness	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.95	**1.00**	0.95	0.81	0.76	**1.00**	**1.00**	0.88
	Precision	**0.97**	**0.98**	0.92	0.95	0.85	0.93	0.93	0.95	**0.98**	**0.97**	0.93	**0.97**	**0.97**	0.88	**0.95**	0.94	0.83
	F-Score	**0.99**	**0.99**	**0.96**	**0.98**	0.92	0.97	0.96	**0.97**	**0.99**	0.96	0.97	0.96	0.88	0.82	**0.98**	0.97	0.86
Ours	**Similarity**	**0.86**	**0.92**	**0.97**	**0.95**	**0.93**	**0.90**	**0.88**	**0.92**	**0.94**	**0.93**	**0.95**	**0.86**	**0.90**	**0.87**	**0.91**	**0.97**	**0.91**
	Fitness	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.94	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	Precision	**0.97**	**0.98**	**0.92**	**0.96**	**0.97**	**0.98**	**0.98**	**0.97**	**0.98**	**0.97**	**0.97**	0.95	**0.97**	**0.96**	**0.95**	**0.98**	**0.95**
	F-Score	**0.99**	**0.99**	**0.96**	**0.98**	**0.99**	**0.99**	**0.99**	0.95	**0.99**	**0.99**	**0.99**	**0.98**	**0.98**	**0.98**	**0.98**	**0.99**	**0.98**

In summary, despite the relative simplicity of the introduced changes, the two baseline approaches generate models that are much more distant from the original model, yet less accurate in capturing the log behavior, than the models produced by our approach.

5.2 Experiment with real-life dataset

In the second experiment, we used a real-life model-log pair of a process for managing road traffic fines in Italy. The normative process model, available as a Petri net (see Fig. 11), is obtained from a textual description of this process [17]. The log[6] is extracted from the information system of a municipality. It contains 150,370 traces of which 231 are distinct and a total of 561,470 events. This log contains a number of anomalies, presumably due to noise and other factors. Examples are traces $\langle \textit{Create fine} \rightarrow \textit{End} \rangle$ and $\langle \textit{Create Fine} \rightarrow \textit{Send Fine} \rightarrow \textit{End} \rangle$, which cannot be replayed in the model.

Covering all log behavior will naturally increase fitness, but at the same time will result in a highly complex and over-fitting model. From the results reported in Table 7, we can see that both baselines cover *all* log behavior (perfect fitness), but result in a very low precision and hence F-Score, and a repaired model that is very different from the original one. In this table, we also report model size as the sum of the number of places and transitions in the Petri net.

These results exacerbate the differences between the three approaches already exposed in Table 6, clearly demonstrating the advantages of our approach over the two baselines. Our approach reduces the fitness of the original model by 0.07, but dramatically increases the precision and, thus the F-score. In addition, it produces a much more readable model (the size is almost half of that of the baselines) which is very close to the

[6] `http://dx.doi.org/10.4121`

original model (the similarity is 0.90 vs. 0.46 for Base1 and 0.55 for Base2). Figure 12 shows the repaired model obtained by our approach, while Fig. 13 shows the model obtained by Base2 (for the sake of comparison, we show the model in Petri nets).

Table 7: Evaluation results on the real-life dataset

	Similarity	**Fitness**	**Precision**	**F-Score**	**Size**
Original model	-	0.99	0.77	0.87	28
Base1 [21]	0.46	**1.00**	0.45	0.62	46
Base2 [10]	0.55	**1.00**	0.49	0.65	50
Ours	**0.90**	0.92	**0.90**	**0.91**	29

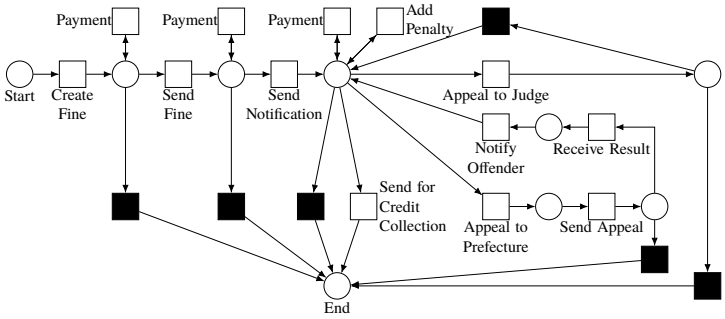


Fig. 11: Normative model of the road traffic fines management process (source: [17])

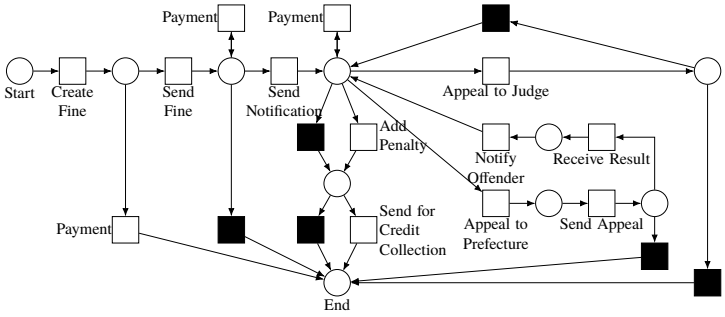


Fig. 12: Our repaired model of the road traffic fines management process

In this second experiment, we only applied the top two differences in terms of number of affected traces, as identified by our tool. Next, we tried a subsequent repair iteration: this increased the precision to 0.93 and the F-Score to 0.92 at the cost of reducing the similarity to 0.86 and increasing the size to 31 nodes. Given that in this dataset the original model is a normative specification, it is up to the user to select which model-log discrepancies to repair based on domain knowledge. In fact, unfitting behavior could be the result of non-compliance, and as such related discrepancies should not be applied to the model, but rather provide opportunities to rectify current practices. Alternatively, they may expose practical workarounds to improve performance, which

could in principle be imported into the normative model. In turn, additional model behavior may point to norms that are ignored in practice, again providing opportunities for rectifying current practices.

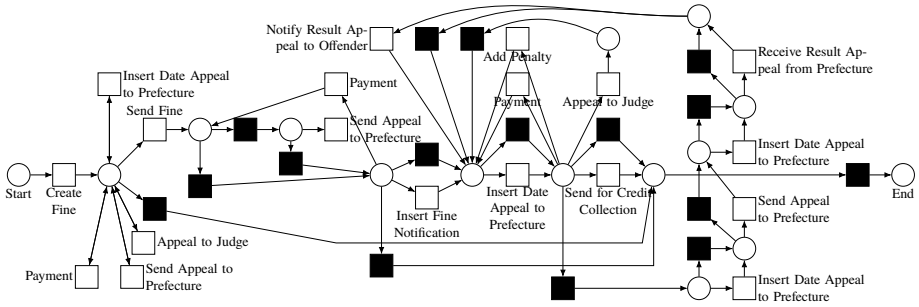


Fig. 13: Process model for the road traffic fines management process repaired by Base2

6 Conclusion

This paper presented a process model repair approach that differs from previous proposals in that it does not seek to fix each discrepancy automatically by adding behavior, but rather, it overlays each discrepancy on top of the model, and lets the user decide incrementally which discrepancies to fix and how, based on the provided visual guidance. Further, the fixes suggested through visual guidance are more accurate (in the sense that they do not add behavior w.r.t. the log), than those provided by state-of-the-art model repair methods. This characteristic is confirmed by the empirical evaluation, which showed that our approach leads to repaired models with a higher F-Score and higher structural similarity relative to two state-of-the-art process model repair methods. The empirical evaluation is limited to a collection of synthetically modified model-log pairs, and one real-life model-log pair. This restricted dataset is a potential threat to the validity of the findings. Conducting a more comprehensive evaluation with further real-life model-log pairs is thus an avenue for future work.

While interactivity is a strength of our approach, it is also a potential limitation insofar as the effort required to repair model-log pairs with many discrepancies may be prohibitive. Another avenue for future work is to automatically identify sets of compatible discrepancies that affect the same model fragment, which can be repaired together in a way that leads to a very similar model with higher F-Score.

Acknowledgments. We thank Artem Polyvyanyy and Raffaele Conforti for their feedback on earlier versions of this work. This research is funded by the Australian Research Council (grant DP150103356) and the Estonian Research Council (grant IUT20-55).

References

1. Adriansyah, A., Muñoz-Gama, J., Carmona, J., van Dongen, B., van der Aalst, W.: Measuring precision of modeled behavior. ISeB 13(1) (2015)
2. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: Proc. of EDOC. IEEE Computer Society (2011)

3. Andrews, K., Wohlfahrt, M., Wurzinger, G.: Visual graph comparison. In: Information Visualisation, 2009 13th International Conference. pp. 62–67. IEEE (2009)
4. Armas-Cervantes, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Diagnosing behavioral differences between business process models: An approach based on event structures. Inf. Syst. 56 (2016)
5. Armas-Cervantes, A., van Beest, N.R.T.P., La Rosa, M., Dumas, M., Raboczi, S.: Incremental and Interactive Business Process Model Repair in Apromore. In: Proc. of BPM Demos. CRC Press (2017 - to appear)
6. van den Brand, M., Protić, Z., Verhoeff, T.: Generic tool for visualization of model differences. In: Proceedings of the 1st international workshop on model comparison in practice. pp. 66–75. ACM (2010)
7. Buijs, J., La Rosa, M., Reijers, H., van Dongen, B., van der Aalst, W.: Improving business process models using observed behavior. In: SIMPDA 2012. Springer (2013)
8. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In: Proc. of BPM. Springer (2009)
9. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer (2013)
10. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. Inf. Syst. 47 (2015)
11. Gambini, M., Rosa, M.L., Migliorini, S., ter Hofstede, A.H.M.: Automated error correction of business process models. In: Proc. of BPM. LNCS, Springer (2011), https://doi.org/10.1007/978-3-642-23059-2_14
12. García-Bañuelos, L., van Beest, N.R., Dumas, M., La Rosa, M.: Complete and interpretable conformance checking of business processes. IEEE Trans. Softw. Eng. (2017), to appear
13. Geyer, M., Kaufmann, M., Krug, R.: Visualizing differences between two large graphs. In: International Symposium on Graph Drawing. pp. 393–394. Springer (2010)
14. Kriglstein, S., Wallner, G., Rinderle-Ma, S.: A visualization approach for difference analysis of process models and instance traffic. In: Business Process Management, pp. 219–226. Springer (2013)
15. La Rosa, M., ter Hofstede, A.H.M., Wohed, P., Reijers, H.A., Mendling, J., van der Aalst, W.M.P.: Managing process model complexity via concrete syntax modifications. IEEE Trans. Industrial Informatics 7(2) (2011)
16. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Fast and accurate business process drift detection. In: Proc. of BPM. Springer (2015)
17. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing 98(4) (2016)
18. Moody, D.: The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Trans. Softw. Eng. 35(6) (2009)
19. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri Nets, Event Structures and Domains, Part I. Theoretical Computer Science 13 (1981)
20. Ohst, D., Welle, M., Kelter, U.: Differences between versions of uml diagrams. In: ACM SIGSOFT Software Engineering Notes. vol. 28, pp. 227–236. ACM (2003)
21. Polyvyanyy, A., van der Aalst, W.M.P., ter Hofstede, A.H.M., Wynn, M.T.: Impact-driven process model repair. ACM Trans. Softw. Eng. Methodol. 25(4) (2016)
22. Rogge-Solti, A., Senderovich, A., Weidlich, M., Mendling, J., Gal, A.: In log and model we trust? a generalized conformance checking framework. In: Proc. of BPM. Springer (2016)
23. van Beest, N., Dumas, M., García-Bañuelos, L., La Rosa, M.: Log delta analysis: Interpretable differencing of business process event logs. In: Proc. of BPM. Springer (2015)
24. van der Aalst, W.: Process Mining: Data Science in Action. Springer (2016)
25. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features: enhancing flexibility in process-aware information systems. Data Knowl. Eng. 66(3) (2008)