Repairing Process Models to Reflect Reality

Dirk Fahland and Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
d.fahland@tue.nl, w.m.p.v.d.aalst@tue.nl

Abstract. Process mining techniques relate observed behavior (i.e., event logs) to modeled behavior (e.g., a BPMN model or a Petri net). Processes models can be discovered from event logs and conformance checking techniques can be used to detect and diagnose differences between observed and modeled behavior. Existing process mining techniques can only uncover these differences, but the actual repair of the model is left to the user and is not supported. In this paper we investigate the problem of *repairing a process model w.r.t. a log* such that the resulting model can replay the log (i.e., conforms to it) and is as similar as possible to the original model. To solve the problem, we use an existing conformance checker that aligns the runs of the given process model to the traces in the log. Based on this information, we decompose the log into several sublogs of non-fitting subtraces. For each sublog, a subprocess is derived that is then added to the original model at the appropriate location. The approach is implemented in the process mining toolkit ProM and has been validated on logs and models from Dutch municipalities.
Keywords: process mining, model repair, Petri nets, conformance checking

1 Introduction

Process mining techniques aim to extract non-trivial and useful information from event logs [1, 14]. Typically three basic types of process mining are considered: (a) *process discovery*, (b) *conformance checking*, and (c) *model enhancement* [1]. The first type of process mining is *process discovery*, i.e., automatically constructing a process model (e.g., a Petri net or a BPMN model) describing the causal dependencies between activities. The basic idea of control-flow discovery is very simple: given an event log containing a set of traces, automatically construct a suitable process model "describing the behavior" seen in the log. However, given the characteristics of real-life event logs, it is notoriously difficult to learn useful process models from such logs. The second type of process mining is *conformance checking* [2, 4, 5, 8, 9, 13, 18, 19, 21, 23]. Here, an existing process model is compared with an event log of the same process. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa. The conformance check could yield that the model *does not describe the process executions observed in reality*: activities in the model are skipped in the log, the log contains events not described by the model, or activities are executed in a different order than described by the model.

Here, the third type of process mining comes into play: to *enhance* an existing model to reflect reality [3]. In principle one could use process discovery to obtain a model

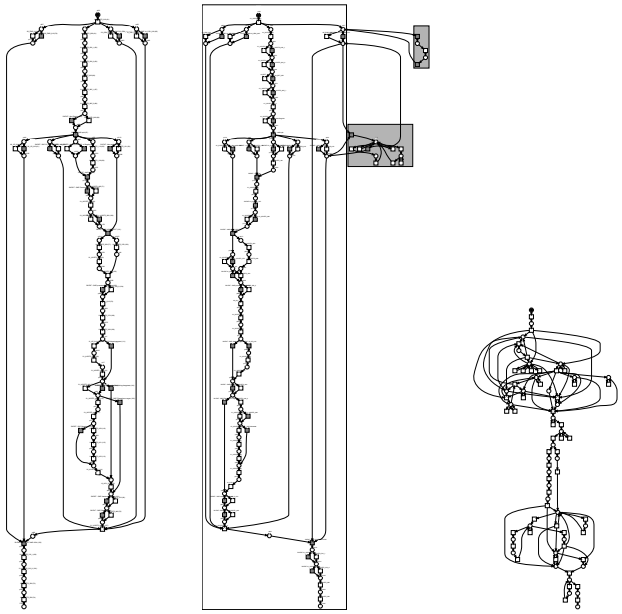

Fig. 1. Original model (left), model (middle) obtained by repairing the original model w.r.t. a given log, and model (right) obtained by rediscovering the process in a new model.

that describes reality. However, the discovered model is likely to bear no similarity with the original model, discarding any value the original model had, in particular if the original was created manually. A typical real-life example is the references process model of a Dutch municipality shown in Fig. 1(left); when rediscovering the actual process using logs from the municipality one would obtain the model in Fig. 1(right). A more promising approach is *repair*, that is change, the original model *so that the repaired model can replay the log and is as similar as possible to the original model*. This is the first paper to focus on model repair with respect to a given log.

The concrete problem addressed in this paper reads as follows. We assume a Petri net N (a model of a process) and a log L (being a multiset of observed cases of that process) to be given. N conforms to L if N can execute each case in L, i.e., N can replay L. If N cannot replay L, then we have to *change* N to a Petri net N' s.t. N' can replay L and N' is as similar to N as possible.

This problem is effectively a *continuum problem* between *confirming that N can replay L* and *discovering a new model N′ from L* in case N has nothing to do with L. The goal is to avoid the latter case as much as possible. We solve this problem in a compositional way: we identify subprocesses that have to be added in order repair N. In more detail, we first compute for each case $l \in L$ an *alignment* that describes at which parts, N and l deviate. Based on this alignment, we identify transitions of N that have to be skipped to replay l and which particular events of l could not be replayed on N. Moreover, we identify the *location* at which N should have had a transition to replay each of these events. We group sequences of non-replayable events at the same location to a *sublog* L' of L. For each sublog L', we construct a small subprocess N'

that can replay L' by using a process mining algorithm. We then insert N' in N at the location where each trace of L' should have occurred. By doing this for every sublog of non-replayable events, we obtain a repaired model that can replay L. Moreover, by the way we repair N, we preserve the structure of N giving process stakeholders useful insights into the way the process changed. We observed in experiments that even in case of significant deviations we could identify relatively few and reasonably structured subprocesses: adding these to the original model always required fewer changes to the original model than a complete rediscovery. Repairing the model of Fig. 1(left) in this way yields the model shown in Fig. 1(middle).

The remainder of this paper is structured as follows. Section 2 recalls basic notions on logs, Petri nets and alignments. Section 3 investigates the model repair problem in more detail. Section 4 presents a solution to model repair based on subprocesses. We report on experimental results in Sect. 5 and discuss related work in Sect. 6. Section 7 concludes the paper.

2 Preliminaries

This section recalls the basic notions on Petri nets and introduces notions such as event logs and alignments.

2.1 Event Logs

Event logs serve as the starting point for process mining. An event log is a multiset of *traces*. Each trace describes the life-cycle of a particular *case* (i.e., a *process instance*) in terms of the *activities* executed.

Definition 1 (Trace, Event Log). *Let Σ be a set of actions. A trace $l \in \Sigma^*$ is a sequence of actions. $L \in \mathbb{B}(\Sigma^*)$ is an event log, i.e., a multiset of traces.*

An event log is a *multiset* of traces because there can be multiple cases having the same trace. If the frequency of traces is irrelevant, we refer to a log as a set of traces $L = \{l_1, \ldots, l_n\}$. In this simple definition of an event log, an event is fully described by an action label. We abstract from extra information such as the resource (i.e., person or device) executing or initiating the activity and the timestamp of the event.

2.2 Petri Nets

We use *labeled* Petri nets to describe processes. We first introduce unlabeled nets and then lift these notions to their labeled variant.

Definition 2 (Petri net). *A Petri net (P, T, F) consists of a set P of* places*, a set T of* transitions *disjoint from P, and a set of arcs $F \subseteq (P \times T) \cup (T \times P)$. A* marking *$m$ of N assigns each place $p \in P$ a natural number $m(p)$ of* tokens*. A* net system *$N = (P, T, F, m_0, m_f)$ is a Petri net (P, T, F) with an* initial *marking m_0 and a final marking m_f.*

We write ${}^\bullet y := \{x \mid (x, y) \in F\}$ and $y^\bullet := \{x \mid (y, x) \in F\}$ for the *pre-* and the *post-set* of y, respectively. Fig. 2 shows a simple net system N with the initial marking $[p_1]$ and final marking $[p_6]$. N will serve as our running example.

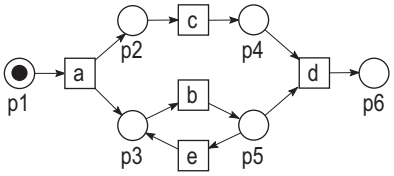

Fig. 2. A net system N.

The semantics of a net system N are typically given by a set of *sequential runs*. A transition t of N is *enabled* at a marking m of N iff $m(p) \geq 1$, for all $p \in {}^\bullet t$. If t is enabled at m, then t may *occur* in the step $m \xrightarrow{t} m_t$ of N that reaches the *successor marking* m_t with $m_t(p) = m(p) - 1$ if $p \in {}^\bullet t \setminus t^\bullet$, $m_t(p) = m(p) + 1$ if $p \in t^\bullet \setminus {}^\bullet t$, and $m_t(p) = m(p)$ otherwise, for each place p of N. A sequential run of N is a sequence $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \ldots \xrightarrow{t_k} m_f$ of steps $m_i \xrightarrow{t_{i+1}} m_{i+1}, i = 0, 1, 2, \ldots$ of N beginning in the initial marking m_0 and ending in the final marking m_f of N. The sequence $t_1 t_2 \ldots t_k$ is an *occurrence sequence* of N. For example, in the net N of Fig. 2 transitions a is enabled at the initial marking; abcd is a possible occurrence sequence of N.

The places and transitions of a Petri net can be *labeled* with names from an alphabet Σ. In particular, we assume label $\tau \in \Sigma$ denoting an *invisible* action. A *labeled Petri net* (P, T, F, ℓ) is a net (P, T, F) with a *labeling* function $\ell : P \cup T \to \Sigma$. A *labeled net system* $N = (P, T, F, \ell, m_0, m_f)$ is a labeled net (P, T, F, ℓ) with initial marking m_0 and final marking m_f. The semantics of a labeled net is the same as for an unlabeled net. Additionally, we can consider *labeled* occurrence sequences of N. Each occurrence sequence $\sigma = t_1 t_2 t_3 \ldots$ of N induces the *labeled* occurrence sequence $\ell(\sigma) = \ell(t_1)\ell(t_2)\ell(t_3)\ldots\ell(t_k)|_{\Sigma\setminus\{\tau\}}$ obtained by replacing each transition t_i by its label $\ell(t_i)$ and omitting all τ's from the result by projection onto $\Sigma \setminus \{\tau\}$. We say that N *can replay* a log L iff each $l \in L$ is a labeled occurrence sequence of N.

2.3 Aligning an Event log to a Process Model

Conformance checking techniques investigate how well an event log $L \in \mathbb{B}(\Sigma^*)$ and a labeled net system $N = (P, T, F, \ell, m_0, m_f)$ fit together. The process model N may have been discovered through process mining or may have been made by hand. In any case, it is interesting to compare the observed example behavior in L and the potential behavior of N. In case the behavior in L is not possible according to N (L cannot replay N), we want to repair N.

In the following we recall a technique for identifying where L and N deviate, and hence where N has to be repaired. It will allow us to determine a *minimal set of changes* that are needed to replay L on N [2, 5, 4]. It essentially boils down to relate $l \in L$ to an occurrence sequence σ of N s.t. l and σ are as similar as possible. When putting l and σ next to each other, i.e., *aligning* σ and l, we will find (1) transitions in σ that are not part of l and (2) activities of l that are not part of σ [2].

For instance, a trace $l = \mathsf{accd}$ is similar to the occurrence sequence $\sigma = \mathsf{abcd}$ of the net of Figure 2 where trace l deviates from σ by skipping over b and having an additional c.

In order to repair N to fit trace l, N has to allow to skip over transitions of the first kind and has to be extended to execute activities of the second kind. In [5, 4] an approach was presented that allows to automatically align a trace l to an occurrence sequence of N with a minimal number of deviations in an efficient way. All of this is based on the notion of an alignment and a cost function.

Definition 3 (Alignment). *Let $N = (P, T, F, \ell, m_0)$ be a labeled net system. Let $l = a_1 a_2 \dots a_m$ be a trace over Σ. A* move *is a pair $(b, s) \in (\Sigma \cup \{\gg\}) \times (T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$. An* alignment *of l to N is a sequence $\alpha = (b_1, s_1)(b_2, s_2) \dots (b_k, s_k)$ of moves, s.t.*

1. *the restriction of the first component to actions Σ is the trace l, i.e., $(b_1 b_2 \dots b_k)|_\Sigma = l$,*
2. *the restriction of the second component to transitions T, $(s_1 s_2 \dots s_k)|_T$, is an occurrence sequence of N, and*
3. *transition labels and actions coincide (whenever both are defined), i.e., for all $i = 1, \dots, k$, if $s_i \neq \gg$, $\ell(s_i) \neq \tau$, and $b_i \neq \gg$, then $\ell(s_i) = b_i$.*

Move (b_i, s_i) is called (1) a *move on model* iff $b_i = \gg \wedge s_i \neq \gg$, (2) a *move on log* iff $b_i \neq \gg \wedge s_i = \gg$, and (3) a *synchronous move* iff $b_i \neq \gg \wedge s_i \neq \gg$.

For instance, for trace $l = \mathsf{accd}$ and the net of Figure 2, a possible alignment would be $(\mathsf{a}, \mathsf{a})(\mathsf{c}, \mathsf{c})(\gg, \mathsf{b})(\mathsf{c}, \gg)(\mathsf{d}, \mathsf{d})$.

Each trace usually has several (possibly infinitely many) alignments to N. We are typically interested in a best alignment, i.e., one that has as many synchronous moves as possible. One way to find a best alignment is to use a cost function on moves and to find an alignment with the least costs.

Definition 4 (Cost function, cost of an alignment). *Let $\kappa : \Sigma \cup T \to \mathbb{N}$ define for each transition and each action a positive cost $\kappa(x) > 1$ for all $x \in \Sigma \cup T$. The* cost *of a move (b, s) is $\kappa(b, s) = 1$ iff $b \neq \gg \neq s$, $\kappa(b, s) = \kappa(s)$ iff $b = \gg$, and $\kappa(b, s) = \kappa(b)$ iff $s = \gg$. The cost of an alignment $\alpha = (b_1, s_1) \dots (b_k, s_k)$ is $\kappa(\alpha) = \sum_{i=1}^{k} \kappa(b_i, s_i)$.*

In this paper, we abstract from concrete cost functions. However, we assume that *desirable* moves, i.e., synchronous moves (b, s) with $\ell(s) = b$ and invisible moves on model $(\gg, s)$ with $\ell(s) = \tau$, have low costs compared to *undesirable* moves such as moves on log $(b, \gg)$ and visible moves on model $(\gg, s)$ with $\ell(s) \neq \tau$.

Definition 5 (Best alignment). *Let $N = (P, T, F, \ell, m_0)$ be a labeled net system. Let κ be a cost function over moves of N and Σ. Let l be a trace over Σ. An alignment α (of l to N) is a* best *alignment (wrt. κ) iff for all alignments α' (of l to N) holds $\kappa(\alpha') \geq \kappa(\alpha)$.*

Note that a trace l can have several best alignments with the same cost. A best alignment α of a trace l can be found efficiently using an A*-based search over the space of all prefixes of all alignments of l. The cost function κ thereby serves as a very efficient heuristics to prune the search space and guide the search to a best alignment. See [5, 4] for details.

Using the notion of best alignment we can relate any trace $l \in L$ to an occurrence sequence of N.

3 Model Repair: The Problem

The model repair problem is to transform a model N that does not conform to a log L into a model N' that conforms to L. We review the state-of-the-art in conformance checking and investigate the model repair problem in more detail.

3.1 Conformance of a Process Model to a Log

Conformance checking can be done for various reasons. First of all, it may be used to audit processes to see whether reality conforms to some normative or descriptive model. Deviations may point to fraud, inefficiencies, and poorly designed or outdated procedures. Second, conformance checking can be used to evaluate the results of process discovery techniques. In fact, genetic process mining algorithms use conformance checking to select the candidate models used to create the next generation of models [17].

Numerous conformance measures have been developed in the past [2, 4, 5, 8, 21, 9, 13, 18, 19, 23]. These can be categorized into four quality dimensions for comparing model and log: (1) *fitness*, (2) *simplicity*, (3) *precision*, and (4) *generalization* [1]. A model with good *fitness* allows for most of the behavior seen in the event log. A model has a perfect fitness if all traces in the log can be replayed by the model from beginning to end. The *simplest* model that can explain the behavior seen in the log is the best model. This principle is known as Occam's Razor. A model is *precise* if it is not "underfitting", i.e., the model does not allow for "too much" behavior. A model is *general* if it is not "overfitting", i.e., the model is likely to be able to explain unseen cases [1, 2].

The fitness of a model N to a log L can be computed using the alignments of Sect. 2.3 as the fraction of moves on log or move on model relative to all moves [2]. The aligned event log can also be used as a starting point to compute other conformance metrics such as precision and generalization.

3.2 Repairing a Process Model to Conform to a Log

Although there are many approaches to compute conformance and to diagnose deviations given a log L and model N, we are not aware of techniques to repair model N to conform to an event log L.

There are two "forces" guiding such repair. First of all, there is the need to improve conformance. Second, there is the desire to clearly relate the repaired model to the original model, i.e., repaired model and original model should be similar. Given metrics for conformance and closeness of models, we can measure the weighted sum or harmonic mean of both metrics to judge the quality of a repaired model. If the first force is weak (i.e., minimizing the distance is more important than improving the conformance), then the repaired model may remain unchanged. If the second force is weak (i.e., improving the conformance is more important than minimizing the distance), then repair can be seen as process discovery. In the latter case, the initial model is irrelevant and it is better to use conventional discovery techniques. Put differently, the model repair problem is positioned in a *spectrum* of two extremes:

keep Keep the original model because it is of high value and non-conformance is within acceptable limits, e.g., 99.9% of all cases can be replayed.

discover Model and log are effectively unrelated to each other, e.g., no case can be replayed and alignments find few or no synchronous moves.

Typically model repair is applied in settings in-between these two extremes. This creates a major challenge: How to identify which parts of a model shall be kept, and which parts of a model shall be considered as nonconformant to the log and hence changed, preferably automatically? The latter is a *local* process discovery problem which requires to balance the four quality dimensions of conformance as well.

3.3 Addressing Different Quality Dimensions

In this paper, we primarily focus on fitness which is often seen as the most important quality dimension for process models. A model that does not fit a given log (i.e., the observed behavior cannot be explained by the model) is repaired using the information available in the alignments.

Only for a fitting model precision can be addressed [18]; our particular technique for model repair will cater for precision as well. The two other criteria of generalization and simplicity may contradict these aims [1, 14]. Generalization and precision can be balanced, for instance using a post-processing technique such as the one presented in [11].

Similarity of the repaired model to the original model, as well as simplicity of the repaired model in general, is harder to achieve. It may require tradeoffs with respect to the other quality dimensions. For model repair basically the same experiences apply as for classical process discovery: while repairing, one should not be forced to extend the model to allow for all observed noisy behavior — it could result in overly complicated, spaghetti-like models. Therefore, we propose the following approach.

1. Given a log L and model N, determine the multiset L_f of fitting traces and the multiset L_n of non-fitting traces.
2. Split the multiset of non-fitting traces L_n into L_d and L_u. According to the domain expert the traces in L_d should fit the model, but do not. Traces in L_u could be considered as outliers/noise (according to the domain expert) and do not trigger repair actions.
3. Repair should be based on the multiset $L' = L_f \cup L_d$ of traces. L' should perfectly fit the repaired model N', but there may be many candidate models N'.
4. Return a repaired model N' that can be easily related back to the original model N, and in which changed parts are structurally simple.

The critical step of separating L_n into L_d and L_u does not require manual inspection of each case by a domain export. A number of standard preprocessing techniques can help to *filter* L_n: by (1) including in L_d only cases with particular start and end events, by (2) filtering infrequent events (that occur only rarely), and by (3) identifying events that occur out of order using *trace alignment* [7]; see [6] for a comprehensive use of preprocessing techniques in a case study.

In the remainder, we assume L' to be given, i.e., outliers L_u of L are removed. If an event log is noisy and one includes also undesired traces L_u, it makes no sense to repair the model while enforcing a perfect fit as the resulting model will be spaghetti-like and not similar to the original model.

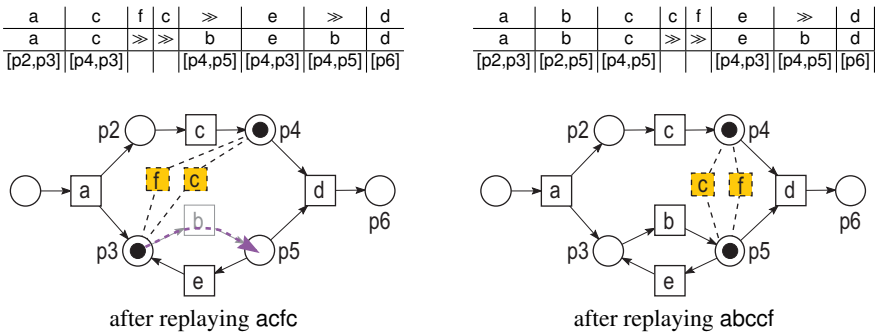

Fig. 3. Alignments of log $L = \{\mathsf{acfced}, \mathsf{abccfed}\}$ to the net of Fig. 2.

4 Repairing Processes by Adding Subprocesses

In the following, we present a solution to model repair. We first sketch a naive approach which completely repairs a model w.r.t. the quality dimension of *fitness* but scores poorly in terms of *precision*. We then define a more advanced approach that also caters for precision. Improvements w.r.t. *simplicity* are discussed at the end.

4.1 Naive Solution to Model Repair – Fitness

Alignments give rise to a naive solution to the model repair problem that we sketch in the following. It basically comprises to extend N with a τ-transition that skips over a transition t whenever there is a move on model $(\gg, t)$, and to extend N with a self-looping transition t with label a whenever there is a move on log $(a, \gg)$. This extension has to be done for all traces and all moves on log/model. The crucial part is to identify the locations of these extensions.

Figure 3 illustrates how the non-fitting log $L = \{\mathsf{acfced}, \mathsf{abccfed}\}$ aligns to the net N of Fig. 2. The nets below each alignment illustrate the differences between log L of Fig. 3 and net N of Fig. 2. After replaying ac, the net is in marking $[\mathsf{p4}, \mathsf{p3}]$ and the log requires to replay f which is not enabled in the net. Thus a log move $(\mathsf{f}, \gg)$ is added. Similarly, c is not enabled at this marking and log move $(\mathsf{c}, \gg)$ is added. Then e should occur, which requires to move the token from $\mathsf{p3}$ to $\mathsf{p5}$, i.e., a model move $(\gg, \mathsf{b})$. Correspondingly, the rest of the alignment, and the second alignment is computed. The third line of the alignment describes the marking that is reached in N by replaying this prefix of the alignment on N.

Using this information, the extension w.r.t. a move on model $(\gg, t)$ is trivial: we just have to create a new τ-labeled transition t^* that has the same pre- and post-places as t. The extension w.r.t. a move on log $(a, \gg)$ provides various options that only require that an a-labeled transition is enabled whenever this move on log occurs. We can use the alignment to identify for each move on log $(a, \gg)$ in which marking m of N it should have occurred (the "enabling location" of this move). In principle, adding an a-labeled transition that consumes from the marked places of m and puts the tokens back immediately, repairs N w.r.t. to this move on log. However, we improve the extension by checking if two moves on log would overlap in their enabling locations. If this is

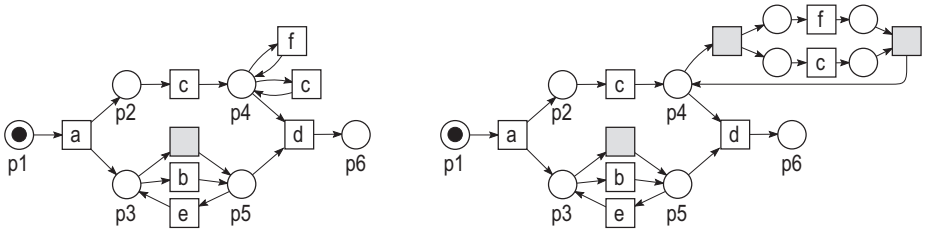

Fig. 4. Result of repairing the net of Fig. 2 w.r.t. the log of Fig. 4 by the naive approach (left) and by adding subprocess (right).

the case, we only add one a-labeled transition that consumes from and produces on this overlap only.

Figure 4(left) shows how model N of Fig. 2 would be repaired w.r.t. the alignment of Fig. 3. The move on model $(\gg, \mathsf{b})$ requires to repair N by adding a τ transition that mimics b as shown in Fig. 4. The move on log $(\mathsf{c}, \gg)$ occurs at two different locations $\{\mathsf{p4}, \mathsf{p3}\}$ and $\{\mathsf{p4}, \mathsf{p5}\}$ in the different traces. They overlap on $\mathsf{p4}$. Thus, we repair N w.r.t. $(\mathsf{c}, \gg)$ by adding a c-labeled transition that consumes from and produces on $\mathsf{p4}$. Correspondingly for $(\mathsf{f}, \gg)$. The extended model that is shown in Fig. 4(left) can replay log L of Fig. 3 without any problems.

4.2 Identify Subprocesses – Precision

The downside of the naive solution to model repair is that the repaired model has low precision. For a log L where a best alignment contains only few synchronous moves, i.e., N does not conform to L, many τ-transitions and self-loops are added. In fact, we observed in experiments that self-looping transitions were often added at the same location creating a "flower sub-process" of events $\Sigma' \subset \Sigma_L$ that locally permitted arbitrary sequences $(\Sigma')^*$ to occur.

In the following, we turn this observation into a structured approach to model repair. Instead of just recording for individual events $a \in \Sigma$ their enabling locations w.r.t. log moves, we now record enabling locations of *sequences of log moves*. Each maximal sequence of log moves (of the same alignment) that all occur at the same location is a *non-fitting sub-trace*. We group non-fitting subtraces at the same location Q into a non-fitting sublog L_Q of that location. We then *discover* from L_Q a *subprocess* $N(L_Q)$ that can replay L_Q by using a mining algorithm that guarantees perfect fitness of $N(L_Q)$ to L_Q. We ensure that $N(L_Q)$ has a unique start transition and a unique end transition. We then add subprocess $N(L_Q)$ to N and let the start transition of $N(L_Q)$ consumes from Q and let the end transition of $N(L_Q)$ produce on Q, i.e., the subprocess models a structured loop that starts and ends at Q.

Figure 4(right) illustrates this idea. The model depicted is the result of repairing N of Fig. 2 by adding subprocesses as described by the alignments of Fig. 3. We can identify two subtraces cf and fc that occur at the same sublocation $\mathsf{p4}$. Applying process discovery on the sublog $\{\mathsf{cf}, \mathsf{fc}\}$ yields the subprocess at the top right of Fig. 4(right) that puts c and f in parallel. The two grey-shaded silent transitions indicate the start and end of this subprocess.

4.3 Formal Definitions

The formal definitions read as follows. For the remainder of this paper, let N be a Petri net system, let L be a log. For each trace $l \in L$, assume an arbitrary but fixed best fitting alignment $\alpha(l)$ to be given. Let $\alpha(L) = \{\alpha(l) \mid l \in L\}$ be the set of all alignments of the traces in L to N.

Locations. Let $\alpha = (a_1, t_1) \dots (a_n, t_n)$ be an alignment w.r.t. $N = (P, T, F, m_0, m_f, \ell)$. For any move (a_i, t_i), let m_i be the marking of N that is reached by the occurrence sequence $t_1 \dots t_{i-1}|_T$ of N. For all $1 \leq i \leq n$, if $(a_i, t_i) = (a_i, \gg)$ is a log move, then the *enabling location* of $(a_i, \gg)$ is the set $loc(a_i, \gg) = \{p \in P \mid m_i(p) > 0\}$ of places that are marked in m_i. For example in Fig. 3, $loc(\mathsf{c}, \gg) = \{\mathsf{p4}, \mathsf{p3}\}$ in the first alignment and $loc(\mathsf{c}, \gg) = \{\mathsf{p4}, \mathsf{p5}\}$ in the second alignment.

It is easy to check that extending N with a new a-labeled transition t with ${}^\bullet t = loc(a_i, \gg) = t^\bullet$ turns the log move $(a_i, \gg)$ into synchronous move (a_i, t), i.e., repairs N w.r.t. $(a_i, \gg)$. We now lift this local repair for one log move to a repair for all alignments of a log to N.

Subtraces. Any two consecutive log moves have the same location as the marking of N does not change. We group these moves into a subtrace. A *maximal* sequence $\beta = (a_i, \gg) \dots (a_{i+k}, \gg)$ of consecutive log moves of α is a *subtrace of α at location Q* iff $loc(a_j, \gg) = loc(a_i, \gg) = Q, i \leq j \leq i + k$, and no longer sequence of log moves has this property. We write $loc(\beta) = loc(a_i, \gg)$ for the location of subtrace β. Let $\beta(L)$ be the set of all subtraces of all alignments $\alpha(L)$ of L to N.

For example, in Fig. 4, fc is a subtrace of the first alignment at location $\{\mathsf{p4}, \mathsf{p3}\}$ and cf is a subtrace of the second alignment at location $\{\mathsf{p4}, \mathsf{p5}\}$. We could repair the net by adding two subprocesses, one that can replay fc at $Q_1 = \{\mathsf{p4}, \mathsf{p3}\}$ and one that can replay cf at $Q_2 = \{\mathsf{p4}, \mathsf{p5}\}$. However, we could instead just add one subprocess that can replay fc *and* cf at location $Q_1 \cap Q_2 = \{\mathsf{p4}\}$.

Formally, we say that Q is a sublocation of a subtrace $\beta = (a_1, \gg) \dots (a_k, \gg)$ iff $Q \subseteq loc(\beta)$. A *sublog* (L_Q, Q) of $\alpha(L)$ at location Q is a set of subtraces $L_Q \subseteq \beta(L)$ s.t. for all $\beta \in L_Q$, $\emptyset \neq Q \subseteq loc(\beta)$, that is, each trace in L_Q can start at sublocation Q of its first event.

Sublogs. The entire set of subtraces $\beta(L)$ can be partitioned into several sublogs of disjoint sublocation, though there are multiple ways of partitioning. We call a set $\{(L_{Q,1}, Q_1), \dots, (L_{Q,k}, Q_k)\}$ of sublogs of $\alpha(L)$ *complete* iff $L_{Q,1} \cup \dots \cup L_{Q,k} = \beta(L)$. While completeness is enough to repair N w.r.t. L, one may want to have as few sublogs at as few locations as possible, for instance, by merging two sublogs $(L_{Q,1}, Q_1)$ and $(L_{Q,2}, Q_2)$ to $(L_{Q,1} \cup L_{Q,2}, Q_1 \cap Q_2)$ if $Q_1 \cap Q_2 \neq \emptyset$. We call $\{(L_{Q,1}, Q_1), \dots, (L_{Q,k}, Q_k)\}$ *minimal* iff $Q_i \cap Q_j = \emptyset$ for all $1 \leq i < j \leq k$. There may be multiple minimal complete sets of sublogs of L. This allows us to configure the repair w.r.t. the locations and the contents of the different sublogs, yielding different repair options.

We now have all notions to formally define how to repair model N w.r.t. log L. For a complete set of sublogs of $\alpha(L)$, we can repair N w.r.t. $\alpha(L)$ by discovering for each sublog (L_Q, Q) a process model N_Q, adding N_Q to N and connecting the start- and end transition of N_Q to Q.

Definition 6 (Subprocess of a sublog). *Let L be a log, let N be a Petri net, let $\alpha(L)$ be an alignment of L to N, and let (L_Q, Q) be a sublog of $\alpha(L)$.*

Let $L_Q^+ = \{start\ a_1 \dots a_k\ end \mid (a_1, \gg) \dots (a_k, \gg) \in L_Q\}$ be the sequences of events described in L_Q extended by a start event and an end event ($start, end \notin \Sigma_L$).

Let $\mathcal{M}$ be a mining algorithm that returns for any log a fitting model (i.e., a Petri net that can replay the log). Let $N_Q = \mathcal{M}(L_Q^+)$. Then (N_Q, Q) is the subprocess *of L_Q.*

The mining algorithm $\mathcal{M}$ will produce transitions labeled with the events in L_Q^+ and a start transition $t_{start}^{N_Q}$ with label *start* and an end transition $t_{end}^{N_Q}$ with label *end*. In the following, we assume that ${}^\bullet t_{start}^{N_Q} = \emptyset$ and $t_{end}^{N_Q\bullet} = \emptyset$, i.e., that start and end transitions have no pre- or post-places. In case $\mathcal{M}$ produced pre- and post-places for start and end, these places can be safely removed without changing that N_Q can replay L_Q^+. When repairing N, we connect $t_{start}^{N_Q}$ and $t_{end}^{N_Q}$ to the location Q of the subprocess.

Definition 7 (Subprocess model repair). *Let L be a log, let N be a Petri net. Let $\alpha(L)$ be the alignments of the traces of L to N.*

Let $\{(L_{Q,1}, Q_1), \dots, (L_{Q,k}, Q_k)\}$ be a minimal and complete set of sublogs of $\alpha(L)$. The subprocess-repaired model *of N w.r.t. $\alpha(L)$ is the net N' that is obtained from N as follows.*

- *Add to N a fresh transition $t_\tau \notin T_N$ with ${}^\bullet t_\tau = {}^\bullet t$ and $t_\tau{}^\bullet = t^\bullet$, $\ell'(t_\tau) = \tau$ iff there exists an alignment $\alpha \in \alpha(L)$ containing a visible model move $(\gg, t)$, and*
- *For each sublog $(L_{Q,i}, Q_i), i = 1, \dots, k$, let $(N_{Q,i}, Q_i)$ be the subprocess of $L_{Q,i}$ s.t. $N_{Q,i}$ and N are disjoint (share no transitions or places). Extend N with $N_{Q,i}$ (add all places, transition and arcs of $N_{Q,i}$ to N) and add arcs $(q, t_{start}^{N_{Q,i}})$ and $(t_{end}^{N_{Q,i}}, q)$ for each $q \in Q_i$, and set labels $\ell'(t_{start}^{N_{Q,i}}) = \tau$ and $\ell'(t_{end}^{N_{Q,i}}) = \tau$.*

Theorem 1. *Let L be a log, let N be a Petri net. Let $\alpha(L)$ be the alignments of the traces of L to N and $\{(L_{Q,1}, Q_1), \dots, (L_{Q,k}, Q_k)\}$ be a minimal and complete set of sublogs of $\alpha(L)$. Let N' be a subprocess-repaired model of N w.r.t. these subtraces. Then each trace $l \in L$ is a labeled occurrence sequence of N', that is, N' can replay L.*

Proof (Sketch). The theorem holds from the observation that each alignment $\alpha = (a_1, t_1) \dots (a_n, t_n) \in \alpha(L)$ of L to N can be transformed into an alignment of L to N' having synchronous moves or model moves on invisible transitions only as follows.

Every move on model $(\gg, t_i)$ of α w.r.t. N is replaced by a model move $(\gg, t_{i,\gg})$ w.r.t. N' on the new invisible transition $t_{i,\gg}$, $\ell(t_{i,\gg}) = \tau$ which allows to skip over t_i.

Every move on log $(a, \gg)$ w.r.t. N is part of a subtrace $\beta = (a_1, \gg) \dots (a_k, \gg)$ of a sublog $(L_{Q,i}, Q_i)$. By adding the subprocess $N_{Q,i}$ at location Q_i, the subtrace β is replayed by a sequence $(\gg, t_{start}^{N_{Q,i}})(a_1, t_1) \dots (a_k, t_k)(\gg, t_{end}^{N_{Q,i}})$ of synchronous moves in the subprocess $N_{Q,i}$. Moves $(t_{start}^{N_{Q,i}}, \gg)$ and $(t_{end}^{N_{Q,i}}, \gg)$ are harmless because they are made silent by relabeling $t_{start}^{N_{Q,i}}$ and $t_{end}^{N_{Q,i}}$ with τ. □

This theorem concludes the techniques for process model repair presented in this paper. Observe that original model N is preserved *entirely* as we only add *new transitions* and *new subprocesses*. By taking a best alignment $\alpha(L)$ of L to N, one ensures that number of new τ-transitions and the number of new subprocesses (or of new self-looping transitions) is minimal.

4.4 Improving Repair – Simplicity

The quality of the model repair step can be improved in some cases. According to Def. 7, each sublog $(L_{Q,i}, Q_i)$ is added as a subprocess $N_{Q,i}$ that consumes from and produces on the same set Q_i of places, i.e., the subprocess is a loop. If this loop is executed in each case of L only once, then $N_{Q,i}$ is also executed exactly once. Thus, N could be repaired by inserting $N_{Q,i}$ in sequence (rather than as a loop), by refining the places $Q_i = \{q_1, \ldots, q_k\}$ to places $\{q_1^-, \ldots, q_k^-\}$ and $\{q_1^+, \ldots, q_k^+\}$ with

1. ${}^\bullet q_j^- = {}^\bullet q_j, q_j^{-\bullet} = \{t_{start}^{N_{Q,i}}\}$, $j = 1 \ldots, k$, and
2. $q_j^{+\bullet} = q_j{}^\bullet, {}^\bullet q_j^+ = \{t_{end}^{N_{Q,i}}\}$, $j = 1 \ldots, k$.

Also, the repaired model N' can structurally be simplified by removing those model elements which are no longer used. Consider for instance a transition t which is never executed because the alignment only includes moves on model $(\gg, t_\tau)$, where t_τ is the new transition to skip over t. In this case t is always bypassed by transition t_τ and t can be removed from N'.

5 Experimental Evaluation

The technique for model repair presented in this paper is implemented in the Process Mining Toolkit ProM 6 in the package, available from `http://www.promtools.org/prom6/`.

Uma provides a plugin *Repair Model* that takes as input a Petri net N, a log L, and a best-fitting alignment $\alpha(L)$ of L to N. The alignment can be computed in ProM 6 using the Conformance Checker of [2, 5, 4]. The *Repair Model* plugin repairs N by extending N with subprocesses as defined in Sect. 4.3 and Sect. 4.4. For this, it first replays each alignment on N, and identifies all subtraces. Then subtraces are grouped to sublogs at the same location. The resulting sublogs are merged if they share the same location in a greedy way (by merging sublogs with the largest overlap of places first), until the resulting set of sublogs is minimal (i.e., all locations are disjoint). Each sublog is then passed to the ILP miner [24] which guarantees to return a model that can replay the sublog. The returned model is then simplified according to [11] and added to N as a subprocess as defined in Def. 7.

We validated our implementation on real-life logs from a process that is shared by five Dutch municipalities. Figure 1(left) shows the reference base model that is used in several municipalities. However, each municipality runs the process differently as demanded by the "couleur locale". As a result, the process observed in each municipality substantially deviates from the base model. To validate our technique, we repaired the base model for each municipality based on the municipality's log. In the following, we report our findings.

We obtained 5 raw logs (M1-M5) from the municipalities' information systems. From these we created filtered logs ($M1^f$-$M5^f$) by removing all cases that clearly should not fit the base model, for instance because they lacked the start of the process or were incomplete (see Sect. 3 for the discussion). Table 1 shows the properties of these 10 logs (over 44 different event classes) discussed in the following. The table lists the number of

Table 1. Results on model repair for 10 logs from Dutch municipalities.

	log		deviations			subprocesses			change to original			
			moves on		per	#	added $\|T\|$		total $\|T\|$		similarity-dist.	
	traces	length	model	log	case		avg.	max.	add.	rem.	repair	discover
M1	434	1-51	3327	310	1-26	7	7	21	69	3	0.144	0.476
M2	286	1-72	1885	323	1-41	5	10	23	65	3	0.147	0.486
M3	481	2-82	3079	1058	1-49	10	13	37	151	3	0.199	0.542
M4	324	1-37	2667	192	2-21	8	7	13	71	4	0.139	0.541
M5	328	2-43	3107	342	2-25	6	9	24	60	3	0.143	0.540
$M1^f$	249	24-40	681	229	1-12	2	6	9	25	4	0.074	0.473
$M2^f$	180	23-70	516	240	1-41	2	12	21	37	5	0.103	0.539
$M3^f$	222	22-82	465	598	1-49	7	10	26	87	5	0.164	0.543
$M4^f$	239	15-37	1216	180	2-17	6	7	13	60	4	0.124	0.542
$M5^f$	328	13-43	1574	280	2-16	4	9	23	51	3	0.111	0.541

traces, minimum and maximum length, and the properties of a best matching alignment of the log to the model of Fig. 1(left) as the total number of model moves, number of log moves and the minimum and maximum number of deviations (log move or model move) per case. None of the traces could be replayed on the base model, in some cases deviations were severe.

Repairing the base model of Fig. 1(left) w.r.t. the filtered log $M1^f$ yields the model of Fig. 1(middle). Repairing the same model w.r.t. the raw log M1 results in the model shown in Fig. 5(left). Repairing the base model w.r.t. the filtered log $M2^f$ yields the model of Fig. 5(right). In each case, model repair requires only several seconds; a best-fitting alignment (needed for repair) could be obtained in about a minute per log. We checked all resulting models for their capability to replay the respective log and could confirm complete fitness for all models.

Moreover, we could re-identify the original model as a sub-structure of the repaired model, making it easy to understand the made repairs in the context of the original model. The original model had 68 transitions, 59 places, and 152 arcs. Table 1 shows for each log the number of added subprocesses, the average and maximal number of transitions per subprocess, and the total number of added and of removed transitions in the entire process. We can see that in the worst case, M3, the number of transitions in the model is more than tripled. Nevertheless, this large number of changes is nicely structured in subprocesses: between 2 and 10 subprocesses were added per log, the largest subprocess had 37 transitions, the average subprocess had 6-13 transitions. We identified alternatives, concurrent actions, and loops in the different subprocesses. Yet, simplification [11] ensured a simple structure in all subprocesses, i.e., graph complexity between 1.0 and 2.0. Model repair also allowed 25%-30% of the original transitions to be skipped by new τ-transitions; only few original transitions could be removed entirely.

To measure similarity, we computed the graph similarity distance [10] between repaired model and original model, and between a completely rediscovered model and the original model. The rediscovered model was obtained with the ILP miner [24] (ensuring fitness) and subsequently simplified by the technique of [11] using the same settings as for subprocess simplification. The similarity distance, roughly, indicates the fraction of

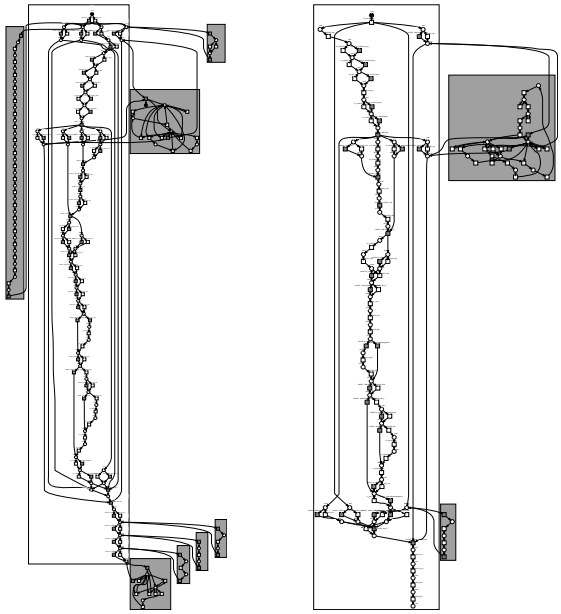

Fig. 5. Result of repairing Fig. 1(left) w.r.t. M1 (left) and $M2^f$ (right).

the original model that has to be changed to obtain the repaired/rediscovered model, i.e., 0.0 means identical models. We observed that original model is significantly more similar to the repaired models (.074-.199) than the original model to the rediscovered models (.473-.543). This indicates that model repair indeed takes the original model structure by far more into account than model repair. The numbers also match the observations one can make when comparing Fig. 1(middle) to Fig. 1(right). Finally, the simpler models and fewer changes required for the filtered logs compared to the raw logs indicate that log preprocessing, as discussed in Sect. 3, has a significant impact on model repair.

6 Related Work

The model repair technique presented in this paper largely relates to two research streams: conformance checking of models and changing models to reach a particular aim.

Various conformance checking techniques that relate a given model to an event log have been developed in the past. Their main aim is to quantify *fitness*, i.e., how much the model can replay the log, and if possible to highlight deviations where possible [2, 4, 5, 8, 21]. The more recent technique of [2, 4] uses alignments to relate log traces to model executions which is a prerequisite for the repair approach presented in this paper. Besides fitness, other metrics [2, 9, 13, 18, 19, 23] (*precision*, *generalization*, and *simplicity*) are used to describe how good a model represents reality. Precision and generalization are currently considered in our approach only as a side-effect and not a leading factor for model repair. Incorporating these measure into model repair is future work. Simplicity is considered in our approach in the sense that changes should be as tractable as possible, which we could validate experimentally.

A different approach to enforcing similarity of repaired model to original model could be model transformation incorporating an edit distance. The work in [15] describes similarity of process model variants based on edit distance. Another approach to model repair is presented in [12] to find for a given model a most similar sound model (using local mutations). [16] considers repairing incorrect service models based on an edit distance. These approaches do not take the behavior in reality into account. Other approaches to adjust a model to reality, adapt the model at runtime [22, 20], i.e., an individual model is created for each process execution. This paper repairs a model for multiple past executions recorded in a log. The approach of [11] uses observed behavior to structurally simplify a given model obtained in process discovery.

7 Conclusion

This paper addressed, for the first time, the problem of repairing a process model w.r.t. a given log. We proposed a repair technique that preserves the original model structure and introduces subprocesses into the model to permit to replay the given log on the repaired model. We validated our technique on real-life event logs and models and showed the approach is effective and the resulting model allows to understand the changes done to the original model for repair.

Our proposed technique of model repair covers the entire problem space of model repair between confirming conformance and complete rediscovery. In case of complete fitness, the model is not changed at all. In case of an entirely unfitting model (no synchronous move), the old model is effectively replaced by a rediscovered model. In case of partial fitness, only the non-fitting parts are rediscovered. This allows to apply our technique also in situations where the given model is understood as a *partial model* (created by hand) that is then completed using process discovery on available logs.

The technique can be configured. The cost-function influences the best-fitting alignment found, grouping of traces into sublogs and identifying sublocations for inserting new subprocesses allows for various solutions. Any process discovery algorithm can be used to discover subprocesses; the concrete choice depends on the concrete conformance notion addressed.

In our future work we would like to consider other conformance metrics such as generalization and precision. Moreover, in our current approach we abstract from extra logging information such as the resource executing or initiating the activity and the timestamp of the event. We would like to incorporate this information when repairing the model. For example, resource information can give valuable clues on for repair.

Acknowledgements. We thank M. Kunze and R.M. Dijkman for providing us with an implementation of the graph similarity distance and the anonymous reviewers for their fruitful suggestions. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n^{o} 257593 (ACSI).

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)

2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Replaying History on Process Models for Conformance Checking and Performance Analysis. WIREs Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
3. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time Prediction Based on Process Mining. Information Systems 36(2), 450–475 (2011)
4. Adriansyah, A., van Dongen, B., van der Aalst, W.M.P.: Conformance Checking using Cost-Based Fitness Analysis. In: EDOC 2011. IEEE Computer Society (2011)
5. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Towards Robust Conformance Checking. In: BPM 2010 Workshops. LNBIP, vol. 66, pp. 122–133 (2011)
6. Bose, R.P.J.C., van der Aalst, W.M.P.: Analysis of Patient Treatment Procedures. In: BPM Workshops'11. LNBIP, vol. 99, pp. 165–166 (2011)
7. Bose, R.P.J.C., van der Aalst, W.M.P.: Process diagnostics using trace alignment: Opportunities, issues, and challenges. Inf. Syst. 37(2), 117–141 (2012)
8. Calders, T., Guenther, C., Pechenizkiy, M., Rozinat, A.: Using Minimum Description Length for Process Mining. In: SAC 2009. pp. 1451–1455. ACM Press (2009)
9. Cook, J.E., Wolf, A.L.: Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. ACM Transactions on Software Engineering and Methodology 8(2), 147–176 (1999)
10. Dijkman, R.M., Dumas, M., García-Bañuelos, L.: Graph Matching Algorithms for Business Process Model Similarity Search. In: BPM. LNCS, vol. 5701, pp. 48–63 (2009)
11. Fahland, D., van der Aalst, W.M.P.: Simplifying Mined Process Models: An Approach Based on Unfoldings. In: Business Process Management (BPM 2011). LNCS, vol. 6896, pp. 362–378. Springer (2011)
12. Gambini, M., Rosa, M.L., Migliorini, S., ter Hofstede, A.H.M.: Automated Error Correction of Business Process Models. In: BPM 2011. LNCS, vol. 6896, pp. 148–165. Springer (2011)
13. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust Process Discovery with Artificial Negative Events. Journal of Machine Learning Research 10, 1305–1340 (2009)
14. IEEE Task Force on Process Mining: Process Mining Manifesto. In: BPM Workshops. LNBIP, vol. 99, pp. 169–194. Springer (2012)
15. Li, C., Reichert, M., Wombacher, A.: Discovering Reference Models by Mining Process Variants Using a Heuristic Approach. In: BPM 2009. LNCS, vol. 5701, pp. 344–362. Springer (2009)
16. Lohmann, N.: Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance. In: BPM 2008. LNCS, vol. 5240, pp. 132–147. Springer (2008)
17. Medeiros, A., Weijters, A., Aalst, W.: Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery 14(2), 245–304 (2007)
18. Munoz-Gama, J., Carmona, J.: A Fresh Look at Precision in Process Conformance. In: BPM 2010. LNCS, vol. 6336, pp. 211–226. Springer (2010)
19. Munoz-Gama, J., Carmona, J.: Enhancing Precision in Process Conformance: Stability, Confidence and Severity. In: CIDM 2011. IEEE, Paris, France (April 2011)
20. Reichert, M., Dadam, P.: ADEPTflex-Supporting Dynamic Changes of Workflows Without Losing Control. JIIS 10(2), 93–129 (March 1998)
21. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems 33(1), 64–95 (2008)
22. Sadiq, S.W., Sadiq, W., Orlowska, M.E.: Pockets of flexibility in workflow specification. In: ER'2001. LNCS, vol. 2224, pp. 513–526 (2001)
23. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A Robust F-measure for Evaluating Discovered Process Models. In: CIDM 2011. pp. 148–155. IEEE (April 2011)
24. van der Werf, J., van Dongen, B., Hurkens, C., Serebrenik, A.: Process Discovery using Integer Linear Programming. Fundamenta Informaticae 94, 387–412 (2010)