ISSN 0146-4116, Automatic Control and Computer Sciences, 2017, Vol. 51, No. 7, pp. 709–723. © Allerton Press, Inc., 2017.
Published in Russian in Modelirovanie i Analiz Informatsionnykh Sistem, 2017, Vol. 24, No. 4, pp. 459–480.

Using Event Logs for Local Correction of Process Models[1]

A. A. Mitsyuk[a,]*, I. A. Lomazova[a,] **, and W. M. P. van der Aalst[b,] ***

[a]*National Research University Higher School of Economics, Laboratory of Process-Aware Information Systems, Moscow, 101000 Russia*

[b]*Eindhoven University of Technology (TU/e), Architecture of Information Systems, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands*

**e-mail: amitsyuk@hse.ru*

***e-mail: ilomazova@hse.ru*

****e-mail: w.m.p.v.d.aalst@tue.nl*

Received July 17, 2017

Abstract—During the life-cycle of an Information System (IS) its actual behavior may not correspond to the original system model. However, to the IS support it is very important to have the latest model that reflects the current system behavior. To correct the model the information from the event log of the system may be used. In this paper, we consider the problem of process model adjustment (correction) using the information from event log. The input data for this task is the initial process model (a Petri net) and an event log. The result of correction should be a new process model, better reflecting the real IS behavior than the initial model. The new model could be also built from scratch, for example, with the help of one of the known algorithms for automatic synthesis of the process model from an event log. However, this may lead to crucial changes in the structure of the original model, and it will be difficult to compare the new model with the initial one, hindering its understanding and analysis. Then it is important to keep the initial structure of the model as much as possible. In this paper we propose a method for process model correction based on the principle of "divide and conquer". The initial model is decomposed into several fragments. For each of the fragments its conformance to the event log is checked. Fragments, which do not match the log, are replaced by newly synthesized ones. The new model is then assembled from the fragments via transition fusion. The experiments demonstrate that our correction algorithm gives good results when it is used for correcting local discrepancies. The paper presents the description of the algorithm, the formal justification for its correctness, as well as the results of experimental testing on artificial examples.

Keywords: process mining, process model repair, Petri nets, process model decomposition, divide and conquer

DOI: 10.3103/S0146411617070306

INTRODUCTION

An event log of a process-aware information system (IS) contains a recorded history of supported processes in the form of a more or less detailed list of events. Most modern ISs record such event logs. The information from event logs can be used for analysis of IS's real behavior and its improvement [3].

In this paper we consider the problem of process model correction (repair) based on the information from an event log. The input data for this problem includes: (1) a process model for correction, and (2) an event log. If the process model, which has been developed earlier, does not fit the current process execution, then the model needs to be corrected. This need arises quite often, since different changes can occur during the life cycle of an information system.

Various formalisms can be used for process modeling. In this work we consider classical Petri net models, with an event log being defined as a multiset of sequences of event records, in which an event record contains information about activity and the timestamp of its execution. As a result of repair the new process model should reflect the IS behavior better than the initial model.

Different process discovery methods based on event logs are described in literature [6, 7, 10, 20, 23, 25, 27, 41, 42]. However, synthesizing a model "from scratch" from a changed event log can lead to a model

[1] The article was translated by the authors.

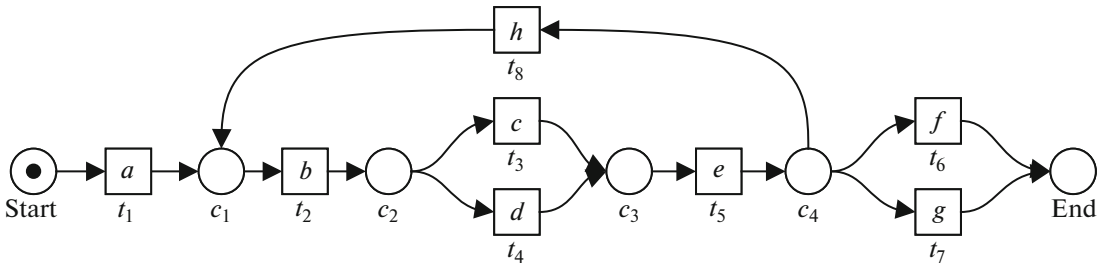

Fig. 1. Example: A compensation requests processing (*transition labels: a = Register request; b = Check ticket; c = Examine casually; d = Examine thoroughly; e = Decide; f = Send rejection letter; g = Pay compensation; h = Re-initiate request*).

which is difficult to match with the initial one. This will complicate its understanding and analysis. Therefore, the model correction problem is to repair the inconsistencies between a model and real behavior of a system, and, as much as possible, save the initial structure of the model.

The general idea of the approach used in this paper is to correct the model using local patch-ups. We propose to decompose the initial model into fragments, select the fragments which do not conform to the event log behavior, and replace them with new repaired fragments. If the changes in the system behavior are local and insignificant, a large part of the model remains unchanged. The general scheme of the approach and its applicability criteria were presented earlier in [28].

This paper has the following structure. Section 1 contains a general description of the model repair problem and an illustrative example. Section 2 gives a short overview of related work. Basic definitions and preliminaries are given in Section 3. Section 4 describes the basic model correction algorithm based on its decomposition into fragments. An enhanced version of the algorithm is presented in Section 5. Section 6 contains an experimental evaluation of the algorithm.

1. PROCESS MODEL REPAIR

We start with a motivating example.

Figure 1 shows a Petri net process model. This example process model from [3] is often used to illustrate methods of process model synthesis. A Petri net in this figure presents processing of expenses compensation requests in some company. Such a model, in particular, can be automatically discovered from the following event log: $[\langle a,b,c,e,f\rangle, \langle a,b,d,e,g\rangle, \langle a,b,c,e,h,b,d,g\rangle]$. This log consists of three traces, each of which is a sequence of events represented by names of executed activities. Time of event execution is used only for event ordering, and can be omitted.

Let us assume, that this event log has been extended with a trace $\langle a,c,b,e,f\rangle$, which can not be replayed by the model. In this new trace an activity b ("Check ticket") is executed after the "Examine casually" activity c. It does not match the order of activities which is allowed by the model in Fig. 1, where all examinations are executed strictly after the ticket check. Thus, the initial model does not conform the *reality*, which is presented for us by the event log.

There are two ways to obtain a model, which reflects the actual process execution: a new model can be discovered from the event log "from scratch," or the existing model can be corrected using an information about the new process behavior. When an initial model is of particular *value* – for example, it has a clear and compact structure – the second approach is much more preferable. Figure 2 shows an example of a possible model repair. Here Most of graphical structure of the original model remains the same. Further we will present the algorithm which will allow to perform such a correction automatically.

Formulation of the Problem of Process Model Repair. Let N be an initial process model in a form of a Petri net, and L be an event log, which contains the current behavior of the process. The problem is to construct a Petri net N^r such that

– N^r can replay all traces from L (the model perfectly fits the log);

– N^r does not allow "too much" executions which are not presented in the log (the model is precise);

– N and N^r have a similar graph structure.

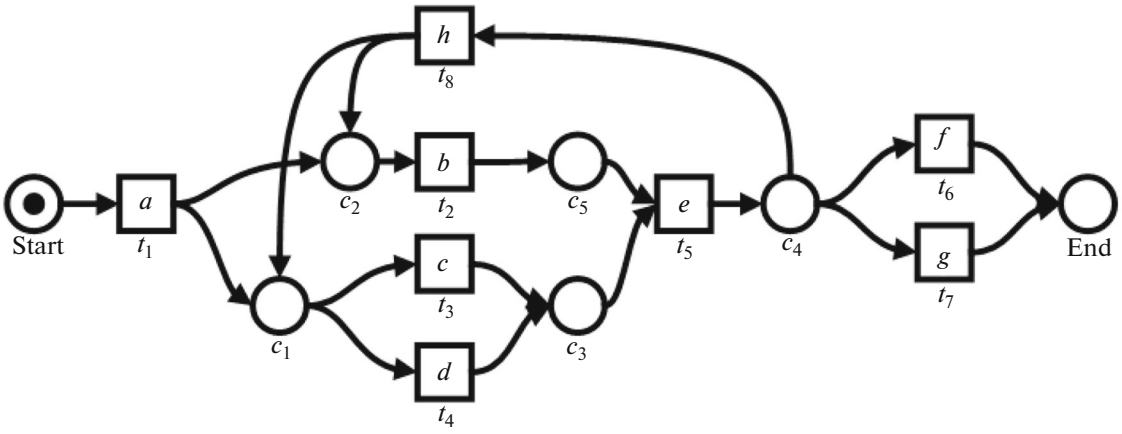

Fig. 2. Example: A corrected model.

2. RELATED WORK

The idea of using information from event logs from process model repair has been proposed in [16] and developed in [17], where authors propose an approach for improving the *fitness* between workflow-net based process model and an event log. To do this, all the traces of event log are split into fitting and non-fitting ones. Non-fitting traces are considered as a separate execution variant for which a sub-process model is discovered, then embedded in the initial model.

Yet another way to repair business-process models, called "Automated Error Correction," has been introduced in [19]. The authors use the method of simulated annealing to find the optimal set of repair operations, and discover one or more workflow nets which model the prescribed process without errors. Note, that an information from event logs is not used here.

A. Polyvyanyy et al. proposed in [31] a method of "impact-driven repair." This method takes into account the impact of various repair operations on some properties of the repaired model. It can be used when the number of permissible repair operations is bounded, and different operations have different effects on the value of the final model. In such a formulation, the Petri net repair problem is reduced to an optimization problem. Authors present the results of a large number of experiments which were conducted to select the most efficient algorithm.

In [12] a method is proposed for improving the well-structured model of process based on a special genetic algorithm, in which, along with four classic quality metrics typical for process mining (fitness, precision, generalization, simplicity), the *similarity* of the discovered model with the original one is taken onto account.

The basic idea of applying the *divide & conquer* principle for process discovery and conformance checking was described in [2, 39]. More practical questions of process model decomposition are considered in [1, 38] and other papers. In particular, these works investigate methods for efficient automatic process model discovery based on event log decomposition.

The *re-composition* problem is considered in [21, 22]. Authors investigate how to choose the most appropriate fragment size for automatic process discovery. Process model and event log decomposition can also be applied for performance improvement in conformance checking [29, 30, 35]. Some modular approaches for process analysis were considered earlier in [5, 26, 33].

The task of the automatic *simplification* of the model is related to the model repair task. Model fragments used in a small number of traces can be removed from it. This simplifies the model with a certain decrease in its fitness to an event log. The methods of simplifying the model are proposed, for example, in [18, 32]. A method for simplification of *fuzzy* process models by removing the non-essential parts of the model using frequency metrics is considered in [20].

The method proposed in this paper differs from the approaches described above. In our approach, the decomposition of the model is used for the subsequent replacement of some fragments. If the discrepancy between the log and the model is local, then the corrected model will differ from the original only by the replaced fragments. This is especially important in cases where the original model was built by an expert, and therefore well perceived by a person. The model, discovered from scratch, can be deprived of such

advantage. Another feature of our approach is that the set of activities is not changed during the repair. This can be a disadvantage (if in reality this set has changed), or an advantage (we do not add any artificial activities).

3. PRELIMINARIES

In this section, the basic definitions and concepts used in the work are given.

Multisets, functions, sequences. By $\mathbb{N}$ we denote the set of natural numbers including zero. Let S be a set. A function $B : S \to \mathbb{N}$, which maps each element of S to the number of its occurrences in B, is a multiset over S. By $\mathcal{B}(S)$ we denote the set of all multisets over S. A finite multiset can be specified by enumerating all its elements and indicating their multiplicity. For example, $B = [e,e,e,c,d,d] = [e^3, c, d^2]$ denotes the multiset with three occurrences of e, one c, and two d. By abuse of notation, we extend the standard set operations to multisets in the usual way.

For a function $f : X \to Y$, $dom(f)$ denotes its domain, and $f : X \nrightarrow Y$ denotes a partial function. A function $f\restriction_Q : Q \nrightarrow Y$ is a projection of a (partial) function f onto a set $Q \subseteq X$ such that $\forall x \in Q : f\restriction_Q(x) = f(x)$. This notation can be extended to multi-sets, e.g. $[e^3, c, d^2]\restriction_{\{c, d\}} = [c, d^2]$.

By X^* we denote the set of all finite sequences over a set X, and we use triangle brackets for sequences, e.g. $\sigma = \langle x_1, x_2, \ldots, x_n \rangle$ is a sequence of length n. By $\sigma_1 \cdot \sigma_2$ we denote the concatenation of two sequences, $\sigma\restriction_Q$ is the projection of the sequence σ onto the set Q.

Process models. In this paper, we use workflow nets (WF-nets) — a subclass of Petri nets — for modeling processes.

A *Petri net* is a triple (P,T,F), where P and T are disjoint sets of *places* and *transitions*, and $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a flow relation. For a transition $t \in T$ a *preset* $\bullet t$ and a *postset* $t\bullet$ are defined as the subsets of P such that $\bullet t = \{p \mid F(p,t) \neq 0\}$ and $t\bullet = \{p \mid F(t,p) \neq 0\}$.

A labeled Petri net is a tuple (P,T,F,l), where $l : T \to \mathbb{U}_A \cup \{\tau\}$ is a labeling function, which maps transitions to activity labels from $\mathbb{U}_A$. A transition t is called invisible, if $l(t) = \tau$, otherwise it is called visible.

A *marking* $M : P \to \mathbb{N}$ specifies a current state of a Petri net. A transition $t \in T$ is *enabled* in a marking M iff $\forall p \in P, M(p) \geq F(p,t)$. An enabled transition t may *fire* yielding a new marking M', such that $M'(p) = M(p) - F(p,t) + F(t,p)$ for each $p \in P$ (denoted $M \xrightarrow{t} M'$).

Definition 1. A *workflow net* (WF-net) is a labeled Petri net $N = (P,T,F,l)$ with distinguished initial M_i and final M_o markings such that

(1) there is one source place $i \in P$ such that $\bullet i = \emptyset$, and $M_i = [i]$,

(2) there is one sink place $o \in P$ such that $o\bullet = \emptyset$, and $M_o = [o]$,

(3) every node $n \in P \cup T$ is on a path from i to o.

Figure 1 shows an example WF-net. Ellipses denote places and squares denote transitions. A black token in the source place denotes the initial marking.

By $T_v(N)$ we denote the set of all visible transitions in N. A transition label is called *unique* if it labels exactly one transition in the net. $T_v^u(N)$ denotes the set of all visible transitions with unique labels in N. The union of two WF-nets is a WF-net $N^U = N^1 \cup N^2$, which is built by the union of sets of places, transitions, and flows: $N^1 \cup N^2 = (P^1 \cup P^2, T^1 \cup T^2, F^1 \cup F^2, l^u)$, where $l^u \in (T^1 \cup T^2) \nrightarrow \mathcal{U}_A$ is a union of l^1 and l^2, $dom(l^u) = dom(l^1) \cup dom(l^2)$, $l^u(t) = l^1(t)$ if $t \in dom(l^1)$, and $l^u(t) = l^2(t)$ if $t \in dom(l^2) \setminus dom(l^1)$. We assume further that all labeled transitions have unique labels.

Event logs. Let $A \subseteq \mathcal{U}_A$ be a set of process *activities*. Since only the names of activities and their sequence are important for constructing the workflow model, we will assume that the *event* in the log is the name of the activity, and we use the terms of the event and of the *activity* (name) as synonyms.

log	a	b	c	e	f
model	a	b	c	e	f
	t_1	t_2	t_3	t_5	t_6

log	a	$\gg$	c	b	e	f
model	a	b	c	$\gg$	e	f
	t_1	t_2	t_3		t_5	t_6

Fig. 3. Alignments (left: perfectly fitting; right: unfitting).

We define a *trace* σ as a finite sequence of activities from A, i.e. $\sigma \in A^*$. An *event log* (or *log*) L is a finite multi-set of traces, i.e. $L \in \mathcal{B}(A^*)$. For example, $L = \left[\langle a,b,c,d,e\rangle^3, \langle a,b,a,d,e\rangle^5, \langle a,d,c,b,e\rangle\right]$ is an event log with the same set of activities as the model shown in Fig. 1.

A projection of an event log $L = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ onto a set of activities B is a log$L\restriction_B = [\sigma_1\restriction_B, \sigma_2\restriction_B, \ldots, \sigma_n\restriction_B]$, where $\sigma_i\restriction_B$ is a projection of the trace σ_i onto B. We call such log a *sub-log* of L.

Conformance checking. The following four criteria are usually used for evaluation of model quality: *fitness*, *precision*, *generalization*, and *simplicity* [3]. The first two are used for checking the conformance between a model and an event log. Fitness shows to what extent the log traces can be replayed by the model. Precision determines to what extent the model can generate behavior that is not represented in the log. Generalization and simplicity characterize the properties of the model itself. Generalization shows the level of model abstraction, and simplicity shows its compactness. A discussion on the meaning and impact of these criteria can be found in [3, 11].

Fitness and precision are of utmost importance during process model repair. Simplicity value is also important, the model should not become too complicated after correction. Besides that, we will measure the similarity between initial and corrected models.

Let N be a workflow net with transition labels from A. Its initial marking is $[i]$ and final marking is $[o]$. Let σ be a trace of A. We say that a trace $\sigma = a_1, \ldots, a_k$ *perfectly fits* a model N, if there is a sequence of transition firings $[i] = m_0 \xrightarrow{t_1} \ldots \xrightarrow{t_k} m_{k+1} = [o]$ such that the sequence of activities $\lambda(t_1), \lambda(t_2), \ldots, da(t_k)$ coincides with σ after the deletion of all silent transitions. An event log L perfectly fits N, if each trace from L perfectly fits N. For example, the set of traces $[\langle a,b,c,e,f\rangle, \langle a,b,c,e,h,b,d,e,g\rangle, \langle a,b,d,e,g\rangle]$ perfectly fits the model in Fig. 1.

A large number of methods for conformance checking are presented in the literature [4, 8, 9, 13, 15, 29, 30, 40]. In this paper, we use a method based on alignments [8]. An *alignment* is a special correspondence between a trace and a sequence of transition firings in a Petri net. Figure 3 shows two examples of an alignment. The top row of each alignment represents the trace from the event log, while the bottom row shows the run in the model (its execution). Each event from a trace must correspond to the transition labeled with the same activity name. If the action in the trace fails to match the corresponding transition in the model execution, then $\gg$ (skip) is added either to the trace or to the model execution.

The left part of Fig. 3 shows the alignment with perfect fitting. In the right alignment we have added two skips, because otherwise this trace can not be replayed by the model.

Obviously, different alignments can be constructed for the same trace-run pair. A good (*optimal*) alignment has the minimal number of skips. Various approaches for constructing optimal alignments, which are used for measuring model to log conformance, are considered in [8].

Process discovery. We apply process discovery methods to repair unfitting model fragments. A lot of such algorithms were described in literature, in particular, in [6, 7, 10, 20, 23–25, 41, 42].

In this paper, we use the *inductive mining* [24] algorithm. This algorithm with certain parameters (we use the setting *inductive miner incompleteness*) guarantees perfect fitness of the resulting model. For testing we additionally use the *ILP Miner*, which allows to reduce the problem of model synthesis to a problem of integer linear programming [41]. This algorithm also guarantees perfect fitness, if executed with suitable parameters.

4. PROCESS MODEL CORRECTION USING DECOMPOSITION

In [28] we have proposed a general modular scheme for process model repair, which uses the *Divide & Conquer* approach. Correcting the process model is divided into the following steps: (1) decomposition,

(2) selection, (3) repair, (4) composition, (5) evaluation. In this paper, we consider a concrete implementation of that scheme, which can be defined as follows.

Let $\mathcal{U}_A$ be a set of acivities, and $\mathcal{U}_N$ be a set of all labeled WF-nets with labels from $\mathcal{U}_A$.

We define the **repair algorithm** as a procedure with an event log $L \in \mathcal{B}\left(\mathcal{U}_A^*\right)$ and a labeled WF-net $N \in \mathcal{U}_N$ at its input, which returns a Petri net $N^r = ModularRepair(L, N)$ perfectly fitting the log L, i.e. $fitness(L, N^r) = 1$, as a result.

The procedural parameters in the modular repair scheme are assigned to:

- $eval \in \left(\mathcal{B}\left(\mathcal{U}_A^*\right) \times \mathcal{U}_N\right) \to [0;1]$ is the evaluation function based on alignments [8],
- $repair \in (\mathcal{B}\left(\mathcal{U}_A^*\right) \times \mathcal{U}_N) \to \mathcal{U}_N$ is the inductive process discovery algorithm [24], or the integer linear programming algorithm [41],
- $split \in \mathcal{U}_N \to \mathcal{P}(\mathcal{U}_N)$ is the algorithm from [28] which constructs the *maximal* decomposition,
- $compose \in \mathcal{P}(\mathcal{U}_N) \to \mathcal{U}_N$ is the composition algorithm based on fusion of transitions with the same labels.

Informally, the repair algorithm can be described as follows. The initial model is decomposed into fragments. For each fragment the algorithm constructs the corresponding sub-log using projection. Then for each pair of the form (a sub-net; a sub-log) their conformance is evaluated. Fragments for which the level of conformance is insufficient, are replaced using some process discovery algorithm. The resulting model is then constructed by combining the fragments along boundary transitions with the same labels.

In [28] we have described the algorithm for the so called 'maximal decomposition' initially proposed in [2]. We use the maximum decomposition for splitting the network into subnets as follows. All nodes in each fragment are divided into boundary (borderline) and internal. Places, invisible transitions, and transitions with non-unique labels can be internal nodes. Only visible transitions with unique – in the initial net – labels can be boundary nodes. A decomposition is called maximal iff the following rule holds for each of the fragments.

Within a fragment, each its internal node can be connected by an arc either with a boundary node, or with another internal node. The boundary node can be connected only with internal nodes of its fragment. Each visible transition with a unique label is divided between two or more fragments (its copy belongs to each of these fragments). An initial marking is distributed between fragment in a natural way: the source place belongs to exactly one fragment with the token residing in it. A maximally decomposed net can be recomposed via fusion of boundary transitions with the same labels. Figure 5 shows a maximal decomposition of the model shown in Fig. 1.

The most important property of maximal decomposition is that it is a emph valid decomposition. The notion of valid decomposition for Petri nets was defined in [2]. Here we use a slightly modified definition of valid decomposition adapted for WF-nets.

Definition 2. Let $N \in \mathcal{U}_N$ be a WF-net with a labeling function l. We call $D = \left\{N^1, N^2, \ldots, N^n\right\} \subseteq \mathcal{U}_N$ its **valid decomposition** iff for $1 \le i < j \le n$

- $N^i = (P^i, T^i, F^i, l^i)$ is a labeled Petri net,
- $l^i = l\restriction_{T^i}$,
- $P^i \cap P^j = \emptyset$,
- $T^i \cap T^j \subseteq T_v^u(N)$, and
- $N = \bigcup_{1 \le i \le n} N^i$.

We denote by $\mathcal{D}(N)$ the set of all valid decompositions of N.

Note, that for a given net, the maximal decomposition is unique. Moreover, it can be shown that the maximal decomposition is a valid decomposition with the minimum possible size of fragments, i.e. it is *maximal*.

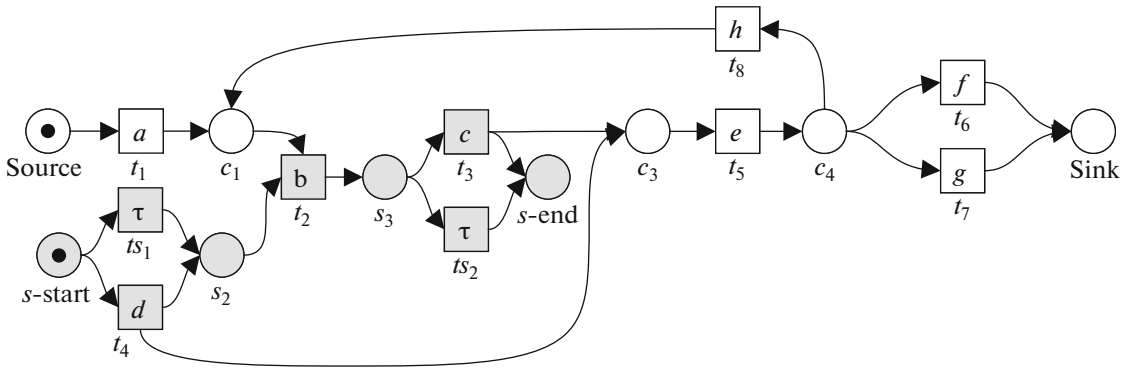

Fig. 4. The repaired model.

It was shown in [2] a valid decomposition can be used for decomposing the fitness checking procedure. Based on these results, in [28] we have formulated the sufficient conditions for ensuring *perfect fitness* of the repaired model.

Proposition 1. The modular repair scheme *ModularRepair* ensures a perfect fitness if

1. *split* is a valid decomposition function;

2. *repair* is a perfect discovery function, i.e. for any event log it returns a Petri net, which perfectly fits it;

3. *compose* is a transition fusion merging all transitions with the same labels.

The proof of this proposition can be found in [28]. Note, that our repair algorithm completely satisfies the conditions (1), (2), and (3). Indeed, we use

(1) maximal decomposition, which is valid,

(2) discovery algorithms, which guarantee perfect fitness by construction,

(3) simple transition fusion for subnets' composition.

Let us consider an example of applying this algorithm. Suppose that we have an event log with traces $\langle a, d, b, e, f \rangle$ and $\langle a, b, c, e, g \rangle$, and it contains no traces in which b precedes d. Thus, d always precedes b, if both appear in the log. Such an event log does not fit the model shown in Fig. 1. In this model, the transition b fires strictly before transitions with labels c and d.

Figure 4 shows the model repaired using our modular approach. This model perfectly fits the log, which consists of traces $\langle a, d, b, e, f \rangle$ and $\langle a, b, c, e, g \rangle$. The repaired model is not a workflow net, since it has additional places *s-start* and *s-end*, which were added by the discovery algorithm. Note, that such a model can be transformed into an equivalent workflow net by adding artificial source and sink places, which should be connected with the places *source* and *s-start* (*sink* and *s-end*) through silent transitions.

Places *s-start* and *s-end* significantly restrict the behavior of the repaired model. In particular, they form a deadlock, which prohibits the execution of the cycle going through the transition t_8. Although this does not contradict the event log, on which the model was corrected, nevertheless, can lower the model value. We remove places *s-start* and *s-end* as redundant to recover this problem. Such an operation does not change fitness between the model and the event log, but it greatly increases the number of possible model runs. Indeed, transitions t_4 and ts_1 can now add tokens to places s_2 and c_3 in each step, whereas the transition ts_2 can remove a token from the place s_3. Therefore, precision of repaired Petri nets may decrease.

In order to cope with this shortcoming, we propose the improved algorithm. It will be described in the next section.

5. IMPROVED ALGORITHM FOR PROCESS MODEL REPAIR

Let us consider the construction of a model for repaired fragment in more details. Primarily, we maximally decompose the model from Fig. 1. Figure 5 shows the result, which consists of six fragments.

Then we calculate the fitness between each fragment and the corresponding log fragment. It turns out that only the fragment $SN3$ does not fit its sub-log. This fragment is replaced by the model synthesized

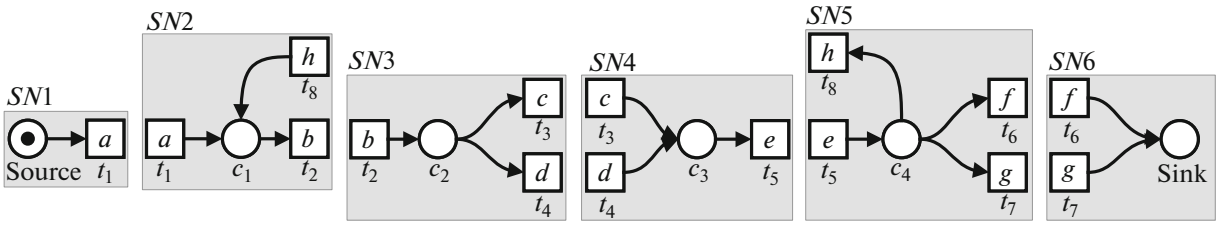

Fig. 5. A maximal decomposition for the model shown in Fig. 1.

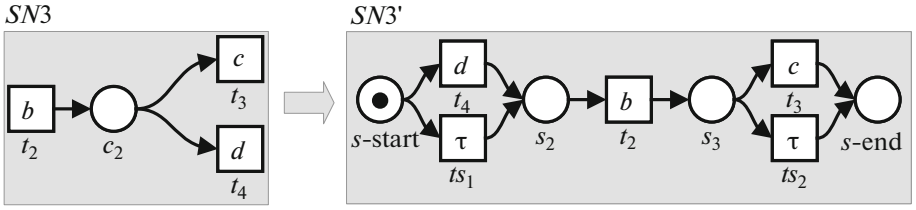

Fig. 6. The replacement of $SN3$ fragment (inductive miner).

using the inductive algorithm (see Fig. 6). Then we compose all fragments into the repaired model which is shown in Fig. 4.

With close examination of the example, it becomes clear that during the repair we also worked with boundary transitions of repaired fragment, over which were the fragments were divided, were also affected. As a result, causal relations on boundary transitions were violated. and this led to a loss of precision of the final model.

We propose to *enlarge* a repaired fragment as a solution to this problem. Such an enlargement should be done according to the certain rule. For each fragment that requires a replacement, it is suggested to attach all adjacent fragments that perfectly fit their sub-logs. As a result of this operation, we can guarantee that outer border of the changed fragment will be not changed during the discovery of a new net.

Indeed, let $m_i \xrightarrow{t_0} \ldots \xrightarrow{t_{n-1}} m_n \xrightarrow{t_n} \ldots \xrightarrow{t_{n+k}} m_{n+k+1} \xrightarrow{t_{n+k+1}} \ldots \xrightarrow{t_{n+k+l}} m_o$ be a sequence of transition firings in a workflow net, going through a large fragment, where transitions $t_n, t_{n+1}, \ldots, t_{n+k-1}, t_{n+k}$ belong to the fragment, t_n and t_{n+k} are boundary transitions, and $t_{n+1}, \ldots, t_{n+k-1}$ are internal transitions. If we attach all neighbor fragments to the changed one, the transitions t_n and t_{n+k} will to belong to one of the neighbor perfectly fitting fragments in the initial decomposition. Therefore, fragments of firing sequences $\ldots \xrightarrow{t_{n-1}} m_n \xrightarrow{t_n} \ldots$ and $\ldots \xrightarrow{t_{n+k}} m_{n+k+1} \xrightarrow{t_{n+k+1}} \ldots$ also fit the event log and they will be not changed by the repairing algorithm.

Figure 7 demonstrates attaching neighbors to the fragment $SN3$ in our example net. This fragment has two neighbors, namely $SN2$ and $SN4$, to be attached.

Definition 3. Let $N^i \in D_N$ be a fragment in some decomposition D of a workflow net N. Neighbour fragments of N^i are all fragments which have common transitions with it. The set of neighbours of N^i is defined as follows: $\mathcal{N}(N^i) = \{N^j \mid N^j \in D_N \wedge T_i \cap T_j \neq \emptyset\}$.

We define an *enlarged* fragment $\mathcal{W}(N^i)$ for N^i as a fragment obtained by attaching to N^i all its neighbors, i.e. $\mathcal{W}(N^i) = N^i \cup \mathcal{N}(N^i)$.

We apply the procedure of fragment enlargement, when there are more than one unfitting fragments in model decomposition, as follows.

Definition 4. Let D be a decomposition of a workflow net N, and $D = D_b \cup D_c$, where D_b is a set of unfitting net fragments, and D_c is a set of other fragments. Obviously, $D_b \cap D_c = \emptyset$. The **fragment enlargement procedure** consists of the following steps:

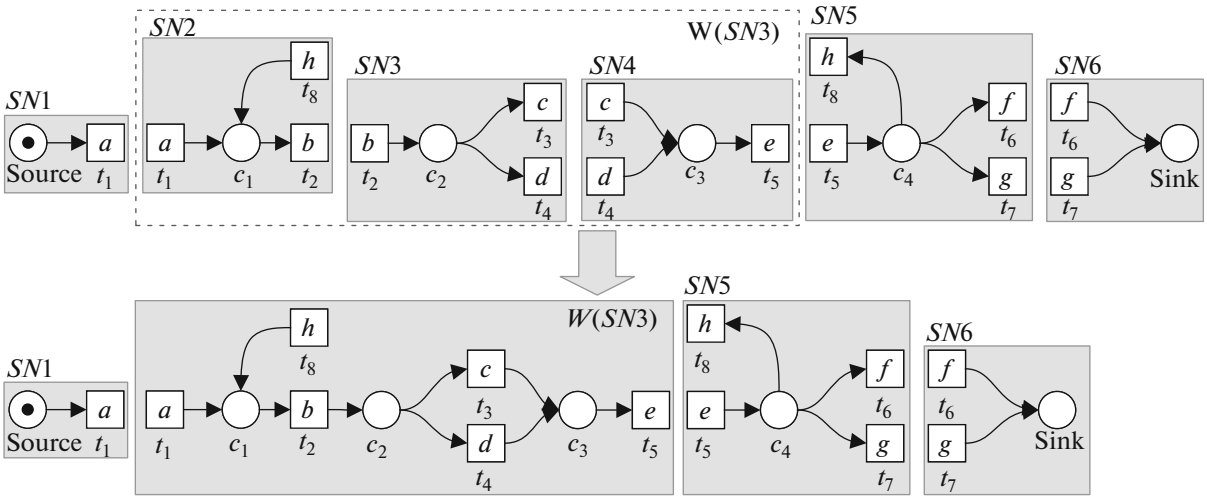

Fig. 7. Neighbours are attached to the fragment.

1. Each fragment from D_b is extended by attaching its neighbors: $\forall N^k \in D_b : D_w^i = \mathcal{W}(N^k)$, $D_n^i = \mathcal{N}(N^k)$; $D_w = \bigcup_i D_w^i$ is a set of all enlarged fragment, and $D_n = \bigcup_i D_n^i$ is a set of all fragments which were affected by the procedure; note, fragments can have common neighbors.

2. All affected fragments are removed from the initial decomposition $D_r = D_c \backslash D_n$.

3. The new decomposition includes enlarged fragments: $D_N^w = D_w \cup D_r$.

Let us show, that the result of fragment enlargement is a valid decomposition.

Proposition 2. Let $D_N = \{N^1, ..., N^n\} \in \mathcal{D}(N)$ be a valid decomposition of a net N. Let D_N^w be an enlarged decomposition in which two fragments are combined, that is $\exists i, j$ such that $1 \le i < j \le n$ and $D_N^w = \{N^i \cup N^j\} \cup D_N \backslash \{N^i, N^j\}$. Then D_N^w is a valid decomposition, i.e. $D_N^w \in \mathcal{D}(N)$.

Proof. We need to show, that all five properties of valid decomposition are valid for $D_N^w = \{N_w^1, ..., N_w^{n-1}\}$.

(1) All subnets $N^1, ..., N^n$ are labeled Petri nets, $N^i \cup N^j$ is also a labeled Petri net by the definition of net union. There is one fragment in a final decomposition, which was constructed by combining two initial fragments, i.e. $\exists k$ such that $1 \le k \le n-1$ and $N_w^k = N^i \cup N^j$. All other fragments were not changed, i.e. $\forall f$ such that $1 \le f \le n-1$ and $f \ne k$: $\exists h$ such that $1 \le h \le n$ and $N_w^f = N^h$.

(2) For all subnets, except the two combined, nothing has changed. When we combine N^i and N^j, we have $dom(l^{N^i \cup N^j}) = dom(l^i) \cup dom(l^j)$ by definition, and $l^{N^i \cup N^j}(t) = l^i(t)$ if $t \in dom(l^i)$, or $l^{N^i \cup N^j}(t) = l^j(t)$ if $t \in dom(l^j) \backslash dom(l^i)$. Thus, $l^k = l\restriction_{T_w^k}$ for any $1 \le k \le n-1$.

(3) Each place remains in its fragment $P^k \cap P^f = \emptyset$ for $1 \le k < f \le n-1$, since in the initial decomposition each place belongs to exactly one fragment, and the set of places for the combined fragments is $P^i \cup P^j$, $i < j$.

(4) Transitions at the fragment boundaries are visible and have unique labels. Indeed, we only decrease the set of boundary transitions, when combine the fragments,

$$\{t \mid t \in T_w^k \cap T_w^f, 1 \le k < f \le n-1\} \subset \{t \mid t \in T^h \cap T^g, 1 \le h < g \le n\} \subseteq T_v^u(N).$$

(5) And finally, directly from the definition of the fragment combining we get: $N = \bigcup_{1 \le h \le n} N^h = \bigcup_{1 \le h < i} N^h \cup \bigcup_{i < h < j} N^h \cup \bigcup_{j < h \le n} N^h \cup N^i \cup N^j = \bigcup_{1 \le f < k} N_w^f \cup \bigcup_{k < f \le n-1} N_w^f \cup N_w^k$.

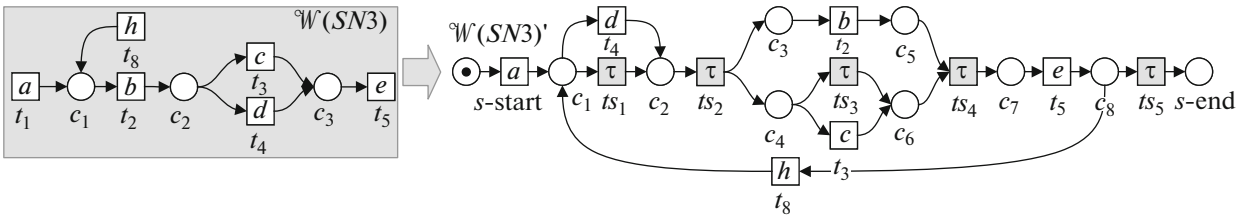

Fig. 8. The replacement of the enlarged fragment $\mathcal{W}(SN3)$.

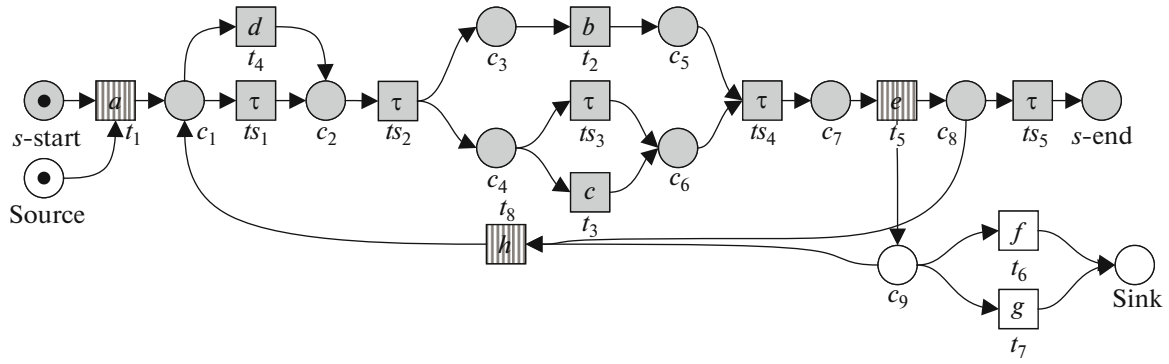

Fig. 9. The repaired model (improved approach).

We have shown that we keep the validity of the decomposition, when we combine two fragments. Combining any number of fragments can be doe by sequential combination of pairs of fragments. Hence, the decomposition with enlarged fragments is valid.

Thus, the basic (*naïve*) repair algorithm can be enhanced by adding one intermediate step. We decompose the initial model, find the unfitting fragments, and enlarge them according to the described procedure. The modified decomposition contains larger fragments. Then we replace unfitting fragments using process discovery algorithm as earlier. We call this procedure the **improved repair algorithm**.

Let us consider an example of applying the improved algorithm. Assume, that we want to repair the model shown in Fig. 1 according to the log which consists of the following traces: $\langle a,d,b,e,f\rangle$, $\langle a,b,c,e,g\rangle$, $\langle a,b,c,e,h,d,b,e,g\rangle$, $\langle a,c,b,e,f\rangle$. We need to replace the fragment $SN3$ of the maximal decomposition, which is shown in Fig. 6. Figure 7 shows how we attach neighbors to this fragment. Then, we project the event log and obtain the sub-log, corresponding to the enlarged fragment: $\langle a,d,b,e\rangle$, $\langle a,b,c,e\rangle$, $\langle a,b,c,e,h,d,b,e\rangle$, $\langle a,c,b,e\rangle$. Figure 8 shows the model, discovered from this sub-log using the inductive discovery algorithm.

Figure 9 shows the model, obtained by combining all fragments. It is easy to verify that this model perfectly fits the log used for repair. Moreover, the model reflects the event log more precisely than the model, which has been discovered without fragment enlargement.

Note, that the places *s-start* and *s-end*, which were added by inductive discovery algorithm, can be also removed from the model. Moreover, the entire fragment consisting of the places c_8, *s-end* and the silent transition ts_5 was added to the fragment during the repair only because the cycle through the transition t_8 turned out to be divided between several fragments.

6. EXPERIMENTAL VALIDATION

We have tested our approach on a number of artificial examples. The test setup has been implemented as a plug-in for *ProM 6 Framework* [37]. The architecture of this software and other details of implementation are described in [34].

The test plan was as follows. First, a perfectly fitting event log for test models was prepared with the use of the generator of event logs [36]. Then, we slightly changed the initial process models in order to spoil

Table 1. Some experimental results *(Similarity is measured with ProM plug-in Calculate Graph Edit Distance Similarity [21], N-f – the number of fragments in decomposition, N-bf – the number of replaced fragments, Size – the number of nodes in the model, Ch. – the number of nodes in changed fragments, Time – the working time in Milliseconds, test configuration: Intel Core i7-3630QM, 2.40 GHz, 4 Gb RAM, Windows 7×64)*

Model	Method	Discovery algorithm	Fitness	Precision	Similarity	N-f	N-bf	Size	Ch.	Time
SM1	Initial model		1	0.91	–	–	–	40	–	–
	New model discovery	Ind	1	0.91	0.51	–	–	47	47	829
		ILP	1	0.91	0.57	–	–	38	38	10069
SM1–BL-1	Changed model		0.94	0.86	–	–	–	40	–	–
	Naive	Ind	1	0.57	0.97	19	1	40	3	2531
		ILP	1	0.57	0.97	19	1	40	3	2531
	Improved	Ind	1	0.91	0.95	19	1	40	7	2365
		ILP	1	0.91	0.95	19	1	40	7	2842
SM1–BL-2	Changed model		0.89	0.89	–	–	–	40	–	–
	Naive	Ind	1	f	0.89	19	3	45	9	2983
		ILP	1	0.5	0.94	19	3	39	9	3215
	Improved	Ind	1	0.91	0.82	19	1	47	15	2426
		ILP	1	0.91	0.88	19	1	41	15	4030
LM2	Initial model		1	0.59	–	–	–	133	–	–
	New model discovery	Ind	1	f	f	–	–	166	166	2920
		ILP	1	0.53	f	–	–	137	137	18116898
LM2–BL-1	Changed model		0.98	0.59	–	–	–	133	–	–
	Naive	Ind	1	0.28	f	63	2	133	7	10745
		ILP	1	0.28	f	63	2	133	7	9588
	Improved	Ind	1	0.59	f	63	1	133	12	9205
		ILP	1	0.59	f	63	1	133	12	10339
LM2–BL-2	Changed model		0.99	0.59	–	–	–	133	–	–
	Naive	Ind	1	0.38	f	63	1	133	3	8048
		ILP	1	0.38	f	63	1	133	3	12847
	Improved	Ind	1	0.59	1	63	1	133	8	12960
		ILP	1	0.59	1	63	1	133	8	8619
LM2–BL-3	Changed model		0.97	0.59	–	–	–	133	–	–
	Naive	Ind	1	0.23	f	63	3	133	10	8097
		ILP	1	0.23	f	63	3	133	10	8855
	Improved	Ind	1	0.59	f	63	2	134	21	9149
		ILP	1	0.59	f	63	2	134	21	12410

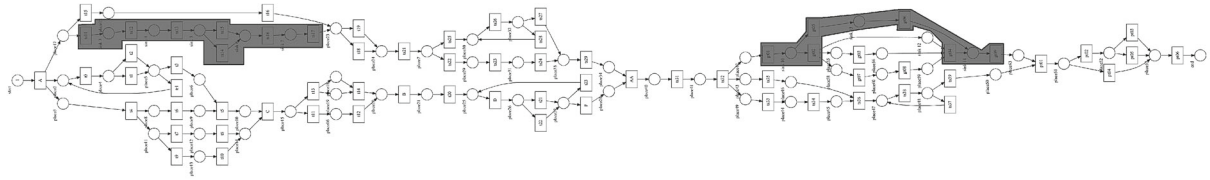

Fig. 10. The repaired model LM2–BL3 (improved approach).

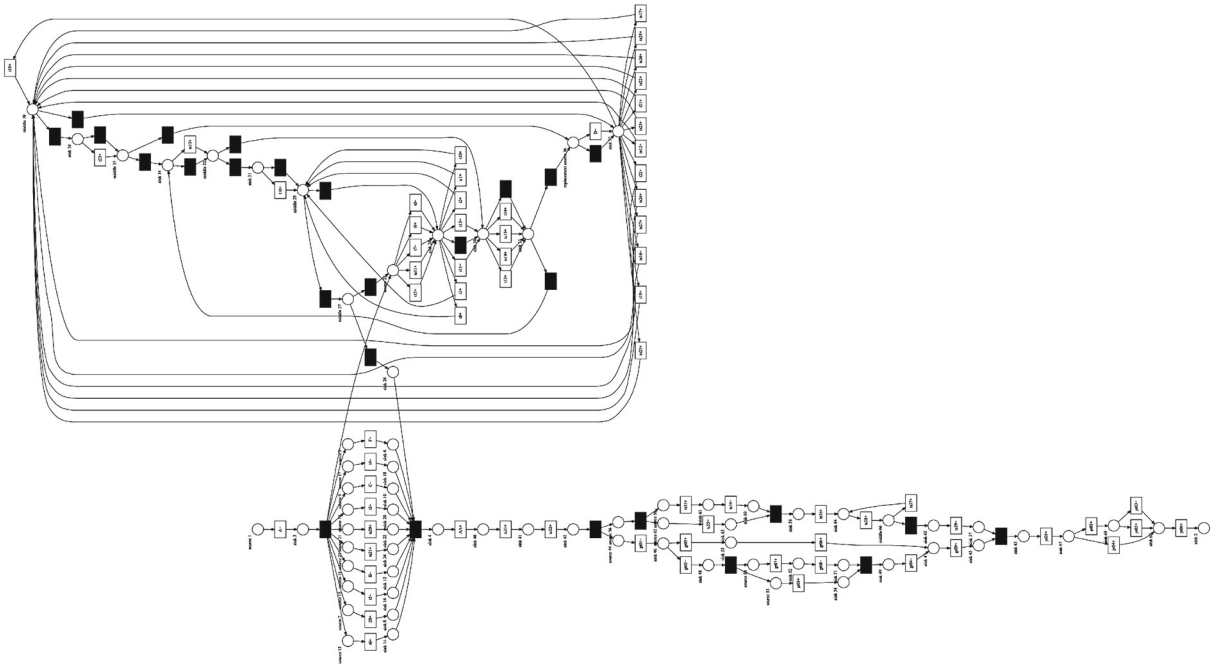

Fig. 11. The model discovered from scratch using the inductive miner from the event log of LM2 model.

their perfect fitting to event logs. The repair algorithm has been tested on these examples. Table 1 shows the obtained results.

We have changed two test workflow nets SM1 and LM2, which consist of 40 and 133 nodes respectively. In each case we replace one or more pairs of transitions in the initial net. Thus, the models -BL1 and so on are obtained. Note that we on purpose change the models not significantly, since our approach is intended for such cases.

Table 1 contains the results of process model repair using the three main approaches: model discovery from scratch (using the log only), naïve and improved model correction methods. We have tested each approach using two algorithms of process discovery, which guarantee perfect fitness of repaired model, namely inductive (Ind) and integer linear programming (ILP) algorithms. Note that our evaluation plug-in failed to calculate metrics in some cases due to high computational complexity. The memory of our machine (4GB) was too small for some cases.

Note also that the results in Table 1 were calculated with removing of source and sink places in replaced fragments, as is described in Section 4.

In this paper, we have described the model repair process using the small model shown in Fig. 1. This is good for reasons of clarity, but the replaced fragment in this case includes most of the model. Note that the effectiveness of our method is higher, the smaller the fragment of the model being changed. Figure 10 shows how the approach deals with the test model LM2–BL3. An affected fragment is highlighted. Figure 11 shows the model, which has been automatically discovered from the event log.

Our algorithm shows the expected results for all examples which are shown in Table 1. In all cases *repaired* models perfectly fit the corresponding logs. The precision of a model decreases when we use the naïve method. Moreover, in some cases the behavior of the model is so diverse that the available 4 Gb of

RAM is not enough for the successful implementation of the algorithm for calculating the precision of the model. With the application of the improved method, the precision for the considered cases can be computed and is close to the precision of the original model.

In all considered examples a changed fragment is a small part of the model. In particular, in the largest example we change 2 of 63 fragments, which together contain 21 transitions and places of the 133 nodes available in the model.

Note, that model repair using ILP algorithm was significantly faster for considered examples, than the automatical synthesis of a completely new model even taking into account the time spent on decomposition and additional fitness checks. This result is expected for such poorly scalable algorithms. A model repair is expectedly slower using inductive algorithm. However, effectiveness optimization is not our goal for now.

Concrete settings of the used process discovery and conformance checking algorithms can change the result model. For example, one can set different costs for nonconformities in alignment-based conformance checking. Thus, some nonconformities can be ignored, if the algorithm is properly configured.

Our method of model correction is not all-purpose. In particular, the problem of non-local repair remains open. A process discovery from the log can be more appropriate than model repair, if changes in a model are significant. Patches only work until a certain point, as for shabby clothes.

CONCLUSIONS

In this paper we have proposed a method of process model correction, which uses an information from the event log. Our method is based on "Divide and Conquer" principle. The model is first divided into fragments with clearly defined borders. Then the fragments, which do not fit this event log, are replaced with new ones, constructed by using some known process discovery algorithms. Then, the repaired model is composed from the fragments. The model obtained is fairly similar to the initial model, and fits the log.

In addition to the formal justification of the method correctness, the work contains the results of testing this method using several artificial examples. These results demonstrate the applicability of the method. It can be especially useful in the cases, when the initial model has been developed by an expert, and thus, is well-understood by a human being. A model, discovered from scratch, can be correct, but unreadable. Our method, however, repairs the model locally and saves its readability.

Nevertheless, the method proposed is not universal. It has proved itself in the case of local inconsistencies, that is we can repair a model by replacing individual fragments. Currently, we are working on improving it to *adjust* the size of the decomposition fragments. However, if the system's behavior has been changed crucially, a process model synthesis from scratch can be more simple and appropriate.

This work continues and specifies the [28], in which we have described the general modular process repair scheme. In particular, here we present one of the several possible implementations of the scheme. In future we plan to consider other potential implementations of the general scheme based on various ways of process model decomposition. The method for process repair, which can add new activities into model, has been described in [17]. We plan to study the possibility of combining this method with the algorithm presented in this paper.

ACKNOWLEDGMENTS

This work is an output of a research project implemented as part of the Basic Research Program at the National Research University Higher School of Economics (HSE).

REFERENCES

1. van der Aalst, W.M.P., Decomposing process mining problems using passages, *Lect. Notes Comput. Sci.,* 2012, vol. 7347, pp. 72–91.
2. van der Aalst, W.M.P., Decomposing Petri Nets for process mining: A generic approach, *Distrib. Parallel Databases,* 2013, vol. 31, no. 4, pp. 471–507.
3. van der Aalst, W.M.P., *Process Mining—Data Science in Action,* Springer, 2016, 2nd ed.
4. van der Aalst, W.M.P., Adriansyah, A., and van Dongen, B.F., Replaying history on process models for conformance checking and performance analysis, *WIREs Data Mining Knowl. Discovery,* 2012, vol. 2, pp. 182–192.
5. van der Aalst, W.M.P., Bolt, A., and van Zelst, S.J., RapidProM: Mine your processes and not just your data, *CoRR,* 2017, abs/1703.03740.

6. van der Aalst, W.M.P., Kalenkova, A.A., Rubin, V.A., and Verbeek, H.M.W., Process discovery using localized events, *Lect. Notes Comput. Sci.,* 2015, vol. 9115, pp. 287–308.
7. van der Aalst, W.M.P., Weijters, A.J.M.M., and Maruster, L., Workow mining: Discovering process models from event logs, *IEEE Trans. Knowl. Data Eng.,* 2004, vol. 16, pp. 1128–1142.
8. Adriansyah, A., Aligning observed and modeled behavior, *PhD Thesis,* Technische Universiteit Eindhoven, 2014.
9. Begicheva, A.K. and Lomazova, I.A., Does your event log fit the high-level process model?, *Model. Anal. Inf. Syst.,* 2015, vol. 22, pp. 392–403.
10. Begicheva, A.K. and Lomazova, I.A., Discovering high-level process models from event logs, *Model. Anal. Inf. Syst.,* 2017, vol. 24, no. 2, pp. 125–140.
11. Buijs, J.C.A.M., van Dongen, B.F., and van der Aalst, W.M.P., On the role of fitness, precision, generalization and simplicity in process discovery, *On the Move to Meaningful Internet Systems: OTM 2012, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10–14, 2012. Proceedings, Part I,* 2012, pp. 305–322.
12. Buijs, J.C.A.M., La Rosa, M., Reijers, H.A., van Dongen, B.F., and van der Aalst, W.M.P., Improving business process models using observed behavior, *Lect. Notes Bus. Inf. Process.,* 2013, vol. 162, pp. 44–59.
13. Burattin, A., Maggi, F.M., and Sperduti, A., Conformance checking based on multi-perspective declarative process models, *Expert Syst. Appl.,* 2016, vol. 65, pp. 194–211.
14. Dijkman, R.M., Dumas, M., and Garcia-Banuelos, L., Graph matching algorithms for business process model similarity search, *Lect. Notes Comput. Sci.,* 2009, vol. 5701, pp. 48–63.
15. van Dongen, B.F., Carmona, J., and Chatain, T., A unified approach for measuring precision and generalization based on anti-alignments, *Lect. Notes Comput. Sci.,* 2016, vol. 9850, pp. 39–56.
16. Fahland, D. and van der Aalst, W.M.P., Repairing process models to reflect reality, *Lect. Notes Comput. Sci.,* 2012, vol. 7481, pp. 229–245.
17. Fahland, D. and van der Aalst, W.M.P., Model repair—aligning process models to reality, *Inf. Syst.,* 2015, vol. 47, pp. 220–243.
18. Fahland, D. and van der Aalst, W.M.P., Simplifying discovered process models in a controlled manner, *Inf. Syst.,* 2013, vol. 38, no. 4, pp. 585–605.
19. Gambini, M., La Rosa M., Migliorini S., and ter Hofstede, A.H.M., Automated error correction of business process models, 9th International Conference Business Process Management (BPM 2011), *Lect. Notes Comput. Sci.,* 2011, vol. 6896, pp. 148–165.
20. Günther, C.W. and van der Aalst, W.M.P., Fuzzy mining: Adaptive process simplification based on multi-perspective metrics, *Lect. Notes Comput. Sci.,* 2007, vol. 4714, pp. 328–343.
21. Hompes, B.F.A., On decomposed process discovery: How to solve a jigsaw puzzle with friends, *Master's Thesis,* Technische Universiteit Eindhoven, 2014.
22. Hompes, B.F.A., Verbeek, H.M.W., and van der Aalst, W.M.P., Finding suitable activity clusters for decomposed process discovery, *4th International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA),* 2015, pp. 32–57.
23. Kalenkova, A.A., Lomazova, I.A., and van der Aalst, W.M.P., Process model discovery: A method based on transition system decomposition, *Lect. Notes Comput. Sci.*, 2014, vol. 8489, pp. 71–90.
24. Leemans, S.J.J., Fahland, D., and van der Aalst, W.M.P., Discovering block-structured process models from incomplete event logs, *Lect. Notes Comput. Sci.*, 2014, vol. 8489, pp. 91–110.
25. Leemans, S.J.J., Fahland, D., and van der Aalst, W.M.P., Scalable process discovery with guarantees, *Lect. Notes Bus. Inf. Process.,* 2015, vol. 214, pp. 85–101.
26. Mans, R., van der Aalst, W.M.P., and Verbeek, H.M.W., Supporting process mining workflows with RapidProM, *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014),* 2014, pp. 56–60.
27. Mitsyuk, A.A. and Shugurov, I.S., On process model synthesis based on event logs with noise, *Autom. Control Comput. Sci.,* 2016, vol. 50, no. 7, pp. 460–470.
28. Mitsyuk, A.A., Lomazova, I.A., Shugurov, I.S., and van der Aalst, W.M.P., Process model repair by detecting unfitting fragments, *Supplementary Proceedings of AIST 2017, CEUR-WS.org,* 2017.
29. Muñoz-Gama, J., Conformance checking and diagnosis in process mining—comparing observed and modeled processes, *Lect. Notes Bus. Inf. Process.,* 2016, vol. 270.
30. Muñoz-Gama, J., Carmona, J., and van der Aalst, W.M.P., Single-entry single-exit decomposed conformance checking, *Inf. Syst.,* 2014, vol. 46, pp. 102–122.
31. Polyvyanyy, A., van der Aalst, W.M.P., ter Hofstede, A.H.M., and Wynn, M.T., Impact-driven process model repair, *ACM Trans. Software Eng. Methodol.,* 2017, vol. 25, no. 4, pp. 28:1–28:60.
32. de San Pedro, J., Carmona, J. and Cortadella, J., Log-based simplification of process models, *Lect. Notes Comput. Sci.*, 2015, vol. 9253, pp. 457–474.

33. Shershakov, S.A., DPMine graphical language for automation of experiments in process mining, *Autom. Control Comput. Sci.,* 2016, vol. 50, no. 7, pp. 477–485.
34. Shugurov, I.S. and Mitsyuk, A.A., Iskra: A tool for process model repair, *Proc. Inst. Syst. Progr. Russ. Acad. Sci.,* 2015, vol. 27, no. 3, pp. 237–254.
35. Shugurov, I.S. and Mitsyuk, A.A., Applying MapReduce to conformance checking, *Proc. Inst. Syst. Progr. Russ. Acad. Sci.,* 2016, vol. 28, no. 3, pp. 103–122.
36. Shugurov, I.S. and Mitsyuk, A.A., Generation of a set of event logs with noise, *Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014),* 2014, pp. 88–95.
37. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., and van der Aalst, W.M.P., ProM 6: The process mining toolkit, *Proc. of BPM Demonstration Track 2010,* 2010, vol. 615, pp. 34–39.
38. Verbeek, H.M.W., Decomposed process mining with divide and conquer, *BPM 2014 Demos,* 2014, vol. 1295, pp. 86–90.
39. Verbeek, H.M.W., van der Aalst, W.M.P., and Muñoz-Gama, J., Divide and conquer: A tool framework for supporting decomposed discovery in process mining, *Comput. J.,* 2017, vol. 1–26.
40. Weber, I., Rogge-Solti, A., Li, C., and Mendling, J., CCaaS: Online conformance checking as a service, *BPM 2015 Demos,* 2015, vol. 1418, pp. 45–49.
41. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., and Serebrenik, A., Process discovery using integer linear programming, *Fundam. Inf.,* 2009, vol. 94, pp. 387–412.
42. Weijters, A.J.M.M., Ribeiro, J., and Chawla, N., Flexible heuristics miner, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011),* 2011, pp. 310–317.