Alexey Alexandrovich Mitsyuk

Structure-Preserving Process Model Repair Based on Event Logs

Ph.D. Dissertation Summary

for the purpose of obtaining

Doctor of Philosophy in Computer Science HSE

Moscow — 2019
This work was prepared at National Research University Higher School of Economics.

Academic Supervisor: **Prof. Dr. Irina A. Lomazova**
Professor, Dr.Sc., Faculty of Computer Science, National Research University Higher School of Economics

Dissertation Committee: **Prof. Dr. Alexander K. Petrenko (chairman),**
Professor, Dr.Sc.,
Software Engineering Department, Institute for System Programming RAS

**Dr. Josep Carmona,**
Associate Professor, Ph.D.,
Department of Computer Science, Universitat Politècnica de Catalunya

**Dr. Marco Montali,**
Associate Professor, Ph.D.,
Faculty of Computer Science, Libera Università di Bozen-Bolzano

**Prof. Dr. Valery A. Sokolov,**
Professor, Dr.Sc.,
Department of Theoretical Information Science, P. G. Demidov Yaroslavl State University

**Prof. Dr. Alessandro Sperduti,**
Professor, Ph.D.,
Dipartimento di Matematica, Università degli Studi di Padova
Dissertation Subject

We can not imagine our life without information systems. Processes in different domains (information technology, banking, healthcare, industrial production etc.) are supported by software systems that store and transform data related to these processes. That is why the concept of process-aware information systems emerged in recent years [1,2]. Systems of that type are based on core models describing process nature.

Complex software and information systems can be designed using model-driven software engineering approaches [3,4]. They involve designing a system structure and processes using formal modelling notations, and then implementing carefully elaborated blueprints in code. However, the exact implementation of a prescriptive design model in real-life systems is a rare case. Moreover, processes tend to evolve and erode during the system’s life-cycle. That is why a description of a real system’s structure and behaviour usually differs from its design model.

System owners want relevant, up-to-date models describing structure and behaviour of their system according to real processes which are executed with this systems’ support. This leads to the development of different methods to reverse engineer a system, analyse its structure and behaviour.

In particular, the real behaviour of a software system can be studied by analysing its event logs. Process mining [5] is a field of research and technology that proposes algorithms and methods for this kind of an analysis. One can discover the model of a real system from event logs. Moreover, process engineers can diagnose the discrepancies between observed (event logs) and modelled (process models) behaviours using conformance checking techniques.

Conformance information can be used to improve or enhance models. For instance, it is possible to repair process model using event logs [6], i.e. to construct a new process model which is based on a given initial model but conforms better to a given event log. Using process model repair techniques a system owner may keep a process model up-to-date. This thesis is devoted to a construction of such algorithms.

Process model repair aims at improving the quality of a model with an additional constraint. The repair should change as few model parts as possible, thus preserving its structure. The latter differs model repair from process discovery, where the goal is to synthesize a model based on the given event log such that this model meets specified conformance characteristics. Thus, model

\[1\] A model quality is calculated according to some quality criteria. In particular, a model needs to conform to a given event log.
repair is applied when existing process model is of value, and its owner does not want to completely re-build it.

The problem can be illustrated with the following example. Figure 1 shows a process model that does not fit the observed behaviour of a system. In particular, its fitness according to the event log is 0.97 (where 1 is perfect fitness).

![Figure 1: Given process model](image1)

We may apply one of process discovery algorithms, and synthesize a completely new model perfectly fitting the same log. For example, Figure 2 shows a model that has been discovered using **Inductive miner** with a zero noise threshold from the event log.

![Figure 2: Model discovered from scratch using the Inductive miner](image2)

This model perfectly fits the event log. It models the same process with the same set of activities, and contains transitions with labels from the same set as the initial model. However, these models are structurally different. Note that the fitness of the initial model is almost perfect. Thus, inconsistencies in it are not serious. Virtually, the model shown in Figure 1 can be repaired replacing two of its elements (transitions). Such problems do not always need the complete re-discovery. Often, they can be repaired without substantial change in the model.

In such a case, structure-preserving algorithms are relevant. This type of model repair is difficult because it needs to find a balance between necessary conformance to an event log, and desire to preserve a structure of the initial model.

---

2This is one of many possible conformance metrics. A model perfectly fits an event log if it can replay all the behaviour recorded in this log. The fitness level shows what portion of log behaviour can be replayed by the model.
model. The goal of this thesis is to propose a structure-preserving process model repair methods.

The problem of this thesis is the model repair problem that can be defined as follows. The initial process model $N$ describes a behaviour of a system. The event log $L$ of this system does not conform to the process model $N$ according to some predefined conformance criteria. In other words, this model $N$ does not fully reflect the observed behaviour of the system. The general model repair problem is to find a model $N^r$ (repaired process model) that conforms the event log $L$ according to the mentioned above criteria, and $N^r$ is as similar to $N$ as possible. Besides, a repair procedure — if possible — does not change the process model significantly.

Main Results of this Thesis

The main research contributions of this thesis are as follows:

I. A new modular approach (scheme) for process model repair based on event logs is presented. It is based on process model decomposition. Process models are represented by workflow nets. The scheme may include various algorithms for model decomposition and sub-nets repair. Sufficient conditions of the approach effectiveness are formulated.

II. New algorithms are presented, performing local and non-local process model repair based on event logs. These algorithms implement the general modular scheme, and employ the divide and conquer principle.

III. New algorithms for event log generation via process model simulation are presented.

IV. Prototype implementations of the process model repair algorithms have been experimentally evaluated using event logs generated by the simulation algorithms.

Publication and Presentation

The results of this thesis have been published in international reviewed journals and conference proceedings. For each paper we provide its status and the proportion of author contribution.

\footnote{Note that in this thesis both following forms are used with the same meaning: a model conforms an event log and an event log conforms a model.}
Publications

Main results of this thesis are published in the following papers which are split in three groups based on their formal status according to the rules of HSE Dissertation Council in Computer Science[4]. In particular, “First-tier publications include papers indexed in the Web of Science (Q1 or Q2) or Scopus (Q1 or Q2) databases, as well as peer-reviewed collections of conferences that appear in CORE rankings (ranks A and A*). Second-tier publications are papers published in journals included in HSE’s list of high quality journals or indexed in the Web of Science (Q3 or Q4) or Scopus (Q3 or Q4) databases, as well as peer-reviewed collections of conferences appearing in CORE rankings (rank B).” Third-tier publications — other papers.

First-tier Publications


Second-tier Publications


Third-tier Publications


Other Papers of the Author

The set of this thesis author’s publications is not limited to the papers listed above. The following papers are related to the subject of process mining and analysis, but contain no of thesis’ contributions. As previously, papers are grouped into three categories.

First-tier Publications

Second-tier Publications


Third-tier Publications


Conferences and Workshops

The results of this thesis have been presented and discussed at the following conferences, seminars, and workshops:


**Thesis Outline**

The thesis consists of the four main chapters, the introduction, and the concluding chapter.

**Introduction** discusses the relevance of thesis subject, informally presents its main problem, contains necessary formal sections, and shortly outlines the thesis content.

The first chapter describes the background of this thesis.
Firstly, Petri nets are defined. This is the language that is used in order to model processes in this thesis. Petri net is a directed bipartite graph with two node types. Transitions model activities of a process, and places are used to specify the order of these activities. In particular, it is possible to model all basic process patterns: sequence, choice, parallel execution, and loops. These nodes are connected with arcs.

The current state of a Petri net is denoted using a so-called marking. A marking is a distribution of tokens through net places. A transition of a Petri net may fire if there are enough tokens in all of its input places. Firing of a transition consumes tokens from its input places and produces tokens to its output places. Thus, a transition firing changes the marking of the net.

In this thesis, workflow nets are used. They form a special class of Petri nets. A workflow net has distinguished initial (source) and final (sink) places. The initial marking of a workflow net is a single token in the source, whereas in the final marking all places are clear except the sink one. All possible runs in such a net start from the initial marking and end in the final marking. Figure 3 shows an example of a workflow net. A black token in the source place denotes the initial marking.

![Figure 3: Simple workflow net](image)

Secondly, event logs are defined. Let $A \subseteq \mathcal{U}_A$ be a set of process activities. They are used to label model transitions. We assume that an event in a log is a name of an activity, i.e. additional event attributes are not used. Thus, an event represents an activity in a trace of an event log.

We define a trace $\sigma$ 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, $L = \{\langle a,b,c,d,e \rangle^3, \langle a,b,a,d,e \rangle^5, \langle a,d,c,b,e \rangle\}$ is an event log with the same activities as in the model shown in Figure 3.

Thirdly, process mining and its three sub-fields are described. Process mining is a research area at the intersection of formal methods and data science [5]. It provides methods for analysis of processes in information systems based on event logs, which are recorded during their lifetime. The three main sub-fields of process mining are described next.
mining are process discovery, conformance checking, and process enhancement. Algorithms from each of these sub-fields are employed in this thesis.

Process discovery algorithms are able to synthesize a process model automatically from a given event log. Conformance checking aims at evaluating the model’s conformance to the observed behaviour that is recorded in an event log. Finally, both processes and process models can be enhanced. Process enhancement provides methods to improve real-life processes according to specified performance criteria. Usually, these methods are less formal, and business-oriented. Process model enhancement methods and techniques to improve models are more formal. The process model repair (or simply model repair) belongs to this category of process mining methods [5].

The four main numerical criteria of a process model are described in this chapter: fitness, precision, generalization, simplicity. These criteria are common in process mining [5].

The first two are used for checking the conformance between a model and an event log. Fitness shows to what extent log traces can be replayed by a model. Precision determines how well the model can generate behaviour 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.

This thesis considers the problem of process model repair according to its fitness. Precision criterion is also employed, when repaired models are evaluated. In this chapter, concrete methods for fitness and precision measurement are described in detail. In this thesis, an alignment-based technique to check log-model conformance [22] is used, since this technique is the most conventional in process mining at the moment.

This chapter also describes the different process discovery algorithms [5]. Two of them are selected to be used in this thesis: Inductive miner [23] and ILP miner [24]. The main reason to use these algorithms with carefully specified settings is that both of them can guarantee perfect fitness of a discovered model.

Fourthly, this chapter also reviews the model repair methods proposed earlier by other researches. Particular problem statements, solutions and their features are considered.

The approach proposed in this thesis has its specific features. The closest approach has been proposed by D. Fahland and W. van der Aalst [25]. However, we employ model decomposition techniques for the subsequent replacement of non-conforming model fragments. Thus, a repaired model differs from the original only by these replaced (repaired) fragments. If discrepancies between an event log and a process model are local, then the change will be insignificant.
It is especially important to preserve a model structure in the cases when the original model has been built by an expert, and therefore this model is well perceived by a human being. The model, re-discovered from scratch using only an event log, can be deprived of such an advantage. Besides, the presented technique is best suited when a set of activities is stable during the repair.

The second chapter presents the main contribution of this thesis: the modular technique for process model repair.

In the first section, we present the specific problem considered in the thesis. Let $A \subseteq \mathcal{U}_A$ be a set of process activities. The initial process model is a workflow net $N = (P, T, F, l)$ with transition labels taken from $\mathcal{U}_A$. The net may have silent transitions labelled with $\tau$. Thus, the labelling function is $l: T \rightarrow A \cup \{\tau\}$.

An event log $L$ is a multi-set of traces, and is defined in Section 1.1.3. Event names are taken from the same set of activities $A$, i.e. $L \in \mathcal{B}(A^*)$. Note that the event names of the log $L$ and the transition labels of the model $N$ are from the set $A$, i.e. $\text{rng}(l) = A$. This means, that there are no new or obsolete activities in the process, and the repair technique does not add/remove transitions to/from the model, except $\tau$-labelled transitions.

Besides, it is assumed that all transitions of a model $N$ except silent transitions have unique labels. That is, for a model $N = (P, T, F, l)$, $\forall t_1, t_2 \in T : l(t_1) = l(t_2)$ then $t_1 = t_2$. Note that this assumption is needed to decompose the process model. In the worst case, if all transitions have the same label, such a model can not be decomposed using decomposition algorithms which are considered in this thesis.

The event log $L$ represents the normative (correct) behaviour that should be modelled, when the model $N$ does not perfectly fit this event log, i.e. $\text{fitness}(L, N) < 1$.

Given a net $N$ and an event log $L$, the problem is to automatically construct a Petri net $N^r$ (repaired model) such that $\text{fitness}(L, N^r) = 1$. Strictly speaking, $L$ perfectly fits $N^r$. In other words, this repaired model $N^r$ can replay all traces of $L$. This is a strict condition of success that needs to be satisfied.

Besides, there are two soft conditions, which are aspired for but may not be satisfied.

Firstly, the precise repaired model $N^r$ is desired. $N^r$ should not allow “too many” execution variants which are not present in $L$. At least, the repaired model $N^r$ should be as precise as the initial model $N$, which means $\text{precision}(N^r) \approx \text{precision}(N)$. In other words, both of these models are approximately equally
precise. Ideally, \( \text{precision}(N^r) = \text{precision}(N) \), but small fluctuations are possible.

Secondly, models \( N \) and \( N^r \) should have a similar structure. The less we change a model by repair, the better this repair algorithm is. That is why a number of model elements touched by a repair procedure reflects the acceptability of a repair in Section 4.3, in which an experimental evaluation of proposed repair technique will be described.

Then, this chapter presents the main contribution of this thesis. Firstly, it presents the modular scheme of the proposed model repair technique. Secondly, algorithms are presented implementing the general modular scheme for particular repair cases.

The modular repair scheme proposed in this thesis employs the idea of patching-up. The algorithm searches non-conforming model fragments, and replaces them with conforming ones. This technique is modular. In other words, the general scheme is constructed of several building blocks. Various particular algorithms can play this role. Thus, the general scheme can be refined into different repair algorithms by choosing appropriate algorithms as its actual procedural parameters.

Each building block of the modular repair scheme executes one repair step. Figure 4 shows a graphical representation of this scheme. Building blocks are highlighted using bold frames.

The input to the scheme is a couple \((L, N)\), where \( L \) is a correct (normative, trusted) event log, and \( N \) is a non-conforming initial model. The model \( N \) is decomposed into several fragments \( N_1, N_2, \ldots, N_k \) using one of the model decomposition algorithms. In the projection block, an algorithm splits the event log into \( k \) sub-logs \( L_1, L_2, \ldots, L_k \) which correspond to \( k \) model fragments (sub-nets). The selection block contains a conformance checking algorithm. This algorithm calculates the conformance value for each pair \((L_i, N_i)\), where \( i = 1, 2, \ldots, k \). According to the corresponding conformance value, all model fragments are separated into two sets. The first one contains conforming (good) fragments. The second one consists of sub-nets which do not conform to corresponding sub-logs (bad fragments). The repair block contains an algorithm which replaces non-conforming model fragments by conforming ones. The composition block is paired with the decomposition one. The result of repair can be evaluated in the evaluation block by using a conformance checking method, or another approach.

This general scheme can be defined more formally, as it is done in the following definition.
**Definition** (Modular Repair Scheme). Let $\mathcal{U}_A$ denote the universal set of activities, and $\mathcal{U}_N$ be the set of all WF-nets with transitions labelled over $\mathcal{U}_A$.

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

The procedural parameters in the modular repair scheme are the following:

- $\text{eval} \in (\mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \to [0; 1]$ is an evaluation function,
- $\text{repair} \in (\mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \to \mathcal{U}_N$ is a repair algorithm,
- $\text{split} \in \mathcal{U}_N \to \mathcal{P}(\mathcal{U}_N)$ is a decomposition algorithm,
- $\text{compose} \in \mathcal{P}(\mathcal{U}_N) \to \mathcal{U}_N$ is a composition algorithm.

Usually, fitness is considered as the main conformance criterion when doing process mining. Indeed, no one needs a model that does not reflect an observed behaviour of the system analysed. Thus, we also consider fitness as the major criterion. Then, we formulate the perfect fitness repair conditions.

**Definition** (Perfect Fitness Repair for Workflow Nets). Let $L \in \mathcal{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$, and let $N \in \mathcal{U}_N$ be a workflow net such that
fitness(\(L, N\)) < 1. Let \(N' = \text{ModularRepair}(L, N)\) be a modular repair scheme. \(\text{ModularRepair}(L, N)\) is a **perfect fitness repair** if fitness(\(L, N'\)) = 1.

The modular repair technique guarantees perfect fitness of a repaired model, if a perfect discovery algorithm is used in the repair building block, and a valid decomposition algorithm is employed to decompose the model. By definition, perfect discovery algorithms construct a perfectly fitting process model for any event log. A decomposition is valid if it successfully decomposes both model and its marking, and allows for an unambiguous composition.

**Theorem** (Perfect Fitness Repair Conditions). *The modular repair scheme \(\text{ModularRepair}(L, N)\) ensures 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 this event log;
3. compose is a transition fusion function that merges all transitions with the same labels.

The proof of this theorem presented in Chapter 2.

Then, we describe the algorithm that employs concrete procedural parameters as building blocks of the general modular scheme.

We assign the procedural parameters in the modular repair scheme as follows:

- evaluation function \(\text{eval} \in (\mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \rightarrow [0; 1]\) is the **alignment-based fitness checking** algorithm [22],
- model repair function \(\text{repair} \in \mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \rightarrow \mathcal{U}_N\) is the **inductive process discovery** algorithm [23], or the **integer-linear-programming-based discovery** algorithm [24],
- decomposition function \(\text{split} \in \mathcal{U}_N \rightarrow \mathcal{P}(\mathcal{U}_N)\) is the **maximal decomposition** algorithm,
- composition function \(\text{compose} \in \mathcal{P}(\mathcal{U}_N) \rightarrow \mathcal{U}_N\) is the algorithm based on **fusion of transitions** with the same labels.

This gives us the **decomposed repair algorithm**. In this chapter, we show that this repair algorithm satisfies the perfect fitness repair conditions 1, 2, and 3.

---

5These algorithms always discover perfectly fitting models.
To implement the repair algorithm, we need to be able to decompose Petri nets maximally. Thus, we propose an algorithm to calculate maximal decomposition of a Petri net. It is presented in Chapter 2.

Figure 5 shows a maximal decomposition of the process model shown in Figure 3. The net is decomposed into six net fragments: $SN_1$, $SN_2$, $SN_3$, $SN_4$, $SN_5$, and $SN_6$. All transitions in the initial model are observable and have unique labels. Thus, transitions $t_1$ (a), $t_2$ (b), $t_3$ (c), $t_4$ (d), $t_5$ (e), $t_6$ (f), $t_7$ (g), and $t_8$ (h) are borderline nodes. Inner nodes are places $source$, $c_1$, $c_2$, $c_3$, $c_4$, and $sink$. One may see how the initial marking is split: the single token moves with the place $source$ in the sub-net $SN_1$. It is easy to see, that fragments can be composed into the workflow net shown in Figure 3 via a fusion of transitions with the same labels.

We call this repair method the naive modular repair technique. Figure 6 shows how the model from Figure 3 can be repaired using this repair algorithm, if the event log consists of two traces: $\langle a, d, b, e, f \rangle$ and $\langle a, b, c, e, g \rangle$.

Repairs of the naive technique are not perfect hence such models are imprecise. That is why the procedure greatly increases the number of possible model runs. The chapter contains detailed discussion of the reasons for this precision reduction. In order to cope with this shortcoming of the naive repair technique, we propose the improved repair algorithm.
The key idea is to enlarge each fragment-to-repair so that the repair procedure will not affect borderline transitions of enlarged fragments. Figure 7 demonstrates attaching neighbours to the fragment $SN_3$ in the example net. This sub-net has two neighbours, namely $SN_2$ and $SN_4$, to be attached.

![Diagram of workflow net](image)

**Figure 7:** Neighbours are attached to the sub-net

We define an *enlarged* fragment $W(N^i)$ for $N^i$ as a fragment obtained by attaching to $N^i$ all its neighbours, i.e. $W(N^i) = N^i \cup \mathcal{N}(N^i)$. Here $\mathcal{N}(N^i) = \{N^j \mid N^j \in D_N \land T_i \cap T_j \neq \emptyset\}$, and $i, j \in \{1, 2, \ldots, k\}$.

The procedure of fragment enlargement is applied when there are more than one unfitting fragment in model decomposition. It can be defined as follows.

**Definition** (Sub-net Enlargement Procedure). Let $D = \{N^1, \ldots, N^n\}$ 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:

1. Each fragment from $D_b$ is extended by attaching its neighbours: $\forall N^i \in D_b$ : $D_i^w = W(N^i)$, $D_i^n = \mathcal{N}(N^i)$; $D_w = \bigcup_i D_i^w$ is a set of all enlarged fragments, and $D_n = \bigcup_i D_i^n$ is a set of all fragments which were affected by the procedure, where $i = 1, 2, \ldots, n$; note that fragments can have common neighbours;

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

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

We have shown that the enlargement procedure can be added to the modular repair technique, because it keeps a decomposition valid. This is formulated in the following theorem (Chapter 2).
Theorem (Sub-net Enlargement Preserves Decomposition Validity). Let $D_N = \{N^1, \ldots, N^n\} \in D(N)$ be a valid decomposition of a net $N$. Let $D^w_N$ be an enlarged decomposition in which two fragments are combined, that is $\exists i, j$ such that $1 \leq i < j \leq n$ and $D^w_N = \{N^i \cup N^j\} \cup D_N \setminus \{N^i, N^j\}$. Then $D^w_N$ is a valid decomposition, i.e. $D^w_N \in D(N)$.

The proof of the theorem is in Chapter 2.

We call this advanced algorithm with a sub-net enlargement the improved modular repair technique. A model repaired using this technique is more precise (see Figure 8) when repairing local inconsistencies.

In a local case, the repair needed for each inconsistency is just a change of the flow relations within a single model fragment. When repairing non-local inconsistencies, we actually need to “move” the transition from one fragment to another, which is impossible using local repair methods. In such a case, naive and improved repair techniques can be used to repair the fitness of a model. However they reduce the values of other conformance metrics, including precision. Figure 9 shows these two types of inconsistencies.

In this chapter, we also describe the greedy repair technique, that is based on the modular repair algorithm, and is able to repair non-local inconsistencies in a desired way.

The input to this algorithm consists of two following components: a workflow net $N$, which will be corrected, and an event log $L$, which contains a real system behaviour. The algorithm decomposes the model using maximal decomposition. Then, it selects the sub-net $N^w_b$ with the worst fitness to the corresponding sub-log. Then, the iteration begins. At each step, a sub-net is enlarged, that is joined with its neighbours $N_b = W(N^i)$, and replaced by the model, which is discovered using a discovery algorithm employed in procedural parameter $\text{discover}(L \upharpoonright_{A(N_b)})$. Then, an intermediate net $N'$ is composed and
Figure 9: Two types of inconsistencies

returned. The algorithm iterates until perfect fitness is achieved between the intermediate net $N'$ and the normative event log $L$. Then, it halts. The model $N'$ is the repaired Petri net.

Figure 10 shows the scheme of this algorithm. In the figure, the key modification of the modular repair scheme is highlighted with light-blue colour.

The example of the greedy repair is shown in Figure 11. The model contains two unfitting fragments which are highlighted with red. In particular, the labels of two transitions were swapped between these fragments. The greedy algorithm started from one of them, and performed nine steps to join the sub-nets. Finally, the whole highlighted part of the model should be replaced with a newly discovered model.

Chapter 2 presents a family of model repair algorithms which is based on the single modular technique. Its main idea is to decompose the initial model, find sub-nets with inconsistencies, and replace them with fitting fragments.

In the practical field of process mining new algorithms should be evaluated using event logs. It is difficult to gather a set of real-life models and event logs with suitable characteristics. Thus, we have developed an approach for generation of synthetic event logs with specified characteristics, which is described in the third chapter.

Using the event log generator, we have prepared a set of suitable event logs and evaluated the modular repair technique. Results of this experimental evaluation are shown in the fourth chapter.

In the third chapter the supportive tools are considered which we use to generate sample event data. We propose a set of algorithms to generate event logs via a simulation of Petri nets and BPMN 2.0 models, and describe the prototype tool that implements presented algorithms.
This chapter consists of two main parts. The first one describes a set of algorithms to generate event logs via a simulation of Petri net models. A similar approach is also applicable for BPMN 2.0 models. This extension is described in the second part of the chapter.

Simulation of a Petri net is based on the standard Petri net firing rule. The main features of the generator tool are as follows:

- Generation settings allow users to decide how many event logs will be generated, and how many traces will these logs include. In order to prevent infinite loops, a user is asked about the maximum number of steps during algorithm execution. All event logs will be generated within single execution of the tool. By default the tool generates 5 event logs while every log consists of 10 traces and it does at most 100 steps.

- In cases of non-deterministic choice, a tool user may specify how the tool selects a transition to fire. A random transition can be fired with equal probability, or preferable transitions can be specified for a choice construct.

- Besides, the tool supports the timed execution of process models. A user can define execution time for every transition and how punctual they are executed.
These fragments do not fit the log

Iterations:

Figure 11: Example of the greedy model repair

by defining deviations bounds. Then, two separate records corresponding
to an event appear in the log. The first one is the start of an activity
execution that is represented as transition firing. The second one is the
termination/completion of an activity.

• The tool allows to create both completely fitting and noisy event logs.

This chapter also describes algorithms to simulate BPMN 2.0 models,
which contain core BPMN elements (tasks, AND-/XOR- gateways), data objects,
and message flows. This approach takes time into account and allows for timed
simulating of process models.

We describe how the proposed simulation approaches have been tested and
evaluated. Using the tool described in this chapter, we have evaluated the modular
repair technique that we presented in the second chapter of this thesis. Results of
this evaluation are considered in the fourth chapter.

The fourth chapter describes a prototype software that implements the
modular repair technique, and presents results of its experimental evaluation.

We have implemented the process model repair approach as a plug-in
for ProM 6 Framework [26] which is a well-known tool in the process mining
community. The main features of this tool and its architecture are described
shortly in this chapter. The prototype software can be found at the page of this
project: http://pais.hse.ru/research/projects/iskra

In order to evaluate our repair techniques, we use the experimental scheme
as shown in Figure [12]. The goal of an experimental evaluation is to apply
a prototype implementation of algorithms to test samples, and measure the
characteristics of result repaired models. These results show pros and cons of
an algorithm when applied to samples with particular properties.
Chapter 4 describes selected process models, and corresponding generated event logs. Each sample model has been used to generate the event log with perfect fitness and other characteristics.

A natural way to test or evaluate the repair technique is as follows. Take a correct (serviceable) object $Obj$, introduce artificial inconsistencies in it ($Obj^b$), and then try to repair it using the evaluated/tested technique. If it works as expected, then the repaired object $Obj^r$ should work (service) as the initial object $Obj$ does. Otherwise, the test is failed. The result of such a repair can be evaluated using suitable measuring tools. Results of different experiments may be compared.

In the fourth chapter, we follow exactly the same way to evaluate the modular process model repair. A repair technique passes the test if it is able to reconstruct the initial sample model. Besides, we applied other repair methods to compare the results. The chapter presents tables and figures which show results of the process model repair using various approaches.

The experiments show an applicability of our approach. The naive technique provides a user with perfectly fitting but significantly less precise models compared to those obtained by using other approaches. Moreover, the behaviour of some models is so diverse that the alignment-based algorithm for calculating model precision lacks available hardware resources. These are cases, when a repaired by the naive technique model contains many silent transitions and transitions without input/output places.
The improved method reconstructed the initial model in most cases. The experimental evaluation shows its effectiveness.

Figure 13 shows the model from Figure 1 repaired using the improved technique. Fragments of the model changed during repair are highlighted with green colour. One may note that such a repair preserves structure of the initial model.

We also evaluated techniques in non-local cases. The greedy technique is effective when a model contains non-local inconsistencies. However, it changes a model more, than the improved technique. One can easily compare naive, improved, and greedy techniques using data shown in this chapter. Within considered examples, the greedy technique constructs models, which are as precise as models discovered from scratch.

Extra large models can also be repaired using the techniques proposed in this thesis. For example, Figure 14 shows a model that contains more than a thousand of nodes which has been repaired using the greedy technique. A fragment that has been replaced by the repair is highlighted with green.

Finally, the last chapter concludes the thesis. Main contributions of this project are summarized in the chapter. It also discusses open issues, and directions for future research on the subject of this thesis.


References


