Automated Repair of Process Models with Non-Local Constraints Using State-Based Region Theory

Anna Kalenkova[*]
School of Computing and Information Systems
The University of Melbourne, Australia
anna.kalenkova@unimelb.edu.au

Josep Carmona[†]
Department of Computer Science, Polytechnic
University of Catalonia, Spain
jcarmona@cs.upc.edu

Artem Polyvyanyy[*]
School of Computing and Information Systems
The University of Melbourne, Australia
artem.polyvyanyy@unimelb.edu.au

Marcello La Rosa[*]
School of Computing and Information Systems
The University of Melbourne, Australia
marcello.larosa@unimelb.edu.au

Abstract. State-of-the-art process discovery methods construct free-choice process models from event logs. Consequently, the constructed models do not take into account indirect dependencies between events. Whenever the input behaviour is not free-choice, these methods fail to provide a precise model. In this paper, we propose a novel approach for enhancing free-choice process models by adding non-free-choice constructs discovered a-posteriori via region-based techniques. This allows us to benefit from the performance of existing process discovery methods and the accuracy of the employed fundamental synthesis techniques. We prove that the proposed approach preserves fitness with respect to the event log while improving the precision when indirect dependencies exist. The approach has been implemented and tested on both synthetic and real-life datasets. The results show its effectiveness in repairing models discovered from event logs.

Keywords: free-choice Petri nets, region state-based synthesis, event logs, transition systems, process mining, process enhancement

1. Introduction

Process mining is a family of methods used for the analysis of event data, e.g., event logs [1]. These methods include *process discovery* aimed at constructing process models from event logs; *conformance checking* applied for finding deviations between real (event logs) and expected (process models)

[*]This work was partly supported by the Australian Research Council Discovery Project DP180102839.
[†]This work was supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

behaviour [2]; and *process enhancement* used for the enrichment of process models with additional data extracted from event logs. The latter also includes *process repair* applied to realign process models in accordance with the event logs. Event logs are usually represented as sequences of events (or traces). The main challenge of process discovery is to efficiently construct *fitting* (capturing traces of the event log), *precise* (not capturing traces not present in the event log) and simple process models.

Scalable process discovery methods, which are most commonly used for the analysis of real-life event data, either produce *directly follows graphs* or use them as an intermediate process representation to obtain a Petri net or a BPMN model [3] (see, e.g., *Inductive miner* [4] and *Split miner* [5]). Directly follows graphs are directed graphs with nodes representing process activities and arcs representing the directly follows (successor) relation between them. Being simple and intuitive, these graphs considerably generalise process behaviour, e.g., they add combinations of process paths that are not observed in the event log. This is because they do not represent higher-level constructs such as parallelism and long-distance (i.e., non-local) dependencies. The above-mentioned discovery methods construct directly follows graphs from event logs and then recursively find relations between sets of nodes in these graphs, in order to discover a *free-choice* Petri net [6], which can then be seamlessly converted into a BPMN model [7] – the industry language for representing business process models. In free-choice nets, the choice between conflicting activities (such that only one of them can be executed) is always "free" from additional preconditions. Although free-choice nets can model parallel activities, non-local choice dependencies are modelled by non-free-choice nets [8]. Several methods for the discovery of non-free-choice Petri nets exist. However, these methods are either computationally expensive [9, 10, 11, 12, 13] or heuristic in nature (i.e., the derived models may fail to replay the traces in the event log) [8]. Other methods are exact and demonstrate good performance, but they usually produce unstructured process models. [14, 15, 16]. In contrast, the approach proposed in this paper starts with a simple free-choice "skeleton", which is then enhanced with additional constraints.

In this paper, we propose a repair approach to enhance free-choice nets by adding extra constructs to capture non-local dependencies. To find non-local dependencies, a transition system constructed from the event log is analysed. This analysis checks whether all the free-choice constructs of the original process model correspond to free-choice relations in the transition system. For process activities with non-free-choice relation in the transition system but with free-choice relation in the process model, region theory [17] is applied to identify, whenever possible, additional places and arcs to be added to the Petri net to ensure the non-local relations between the corresponding transitions. Remarkably, although we have implemented our approach over *state-based* region theory [9, 10, 11], the proposed approach can also be extended to *language-based* region theory [12, 18], or to geometric or graph-based approaches that have been recently proposed [19, 20].

Importantly, we apply a goal-oriented state-based region algorithm to those parts of the transition system where the free-choice property is not fulfilled. This allows us to reduce the computation time, relegating region-theory to when it is needed. We prove that important quality metrics of the initial free-choice (workflow) net are either preserved or improved for those cases where non-local dependencies exist, i.e., *fitness* is never reduced, and *precision* can increase. Hence, when using our approach on top of an automated discovery method that returns a free-choice Petri net, one can still keep the complexity of process discovery manageable, obtaining more precise process models that represent the process behaviour recorded in the event log.

In contrast to the existing process repair techniques, which change the structure of the process models by inserting, removing [21, 22, 23] or replacing tasks and sub-processes [24], the approach proposed in this paper only imposes additional restrictions on the process model behaviour, preserving fitness and improving precision where possible.

We implemented the proposed approach as a plugin of Apromore [25][1] and tested it both on synthetic and real-world event data. The tests show the effectiveness of our approach within reasonable time bounds.

This article is an extended version of our conference paper [26]. It makes the following additions to the original conference paper:

- Extends the repair approach to the set of free-choice models with *silent* transitions;
- Presents an optimized version of the repair algorithm and analyses its time complexity;
- Introduces a technique to convert workflow nets with non-local constraints to high-level BPMN models with data objects;
- Reports on large-scale experiments of applying the proposed techniques to real-world data.

The paper is organized as follows. Section 2 illustrates the approach by a motivating example. Section 3 contains the main definitions used throughout the paper. The state-based region technique is introduced in Section 4. The proposed model repair approach is then described in Section 5. Additionally, Section 5 contains formal proofs of the properties of the repaired process model. High-level process modelling constructs, e.g., BPMN modelling elements representing non-free-choice routing, are also discussed in Section 5. The results of the experiments are presented in Section 6. Finally, Section 7 concludes the paper.

2. Motivating Example

This section presents a simple motivating example inspired by the real-life BPIC'2017 event log[2] and examples discussed in [8]. Consider the process of a loan application. The process can be carried out by a client or by a bank employee on behalf of the client. Thus, this process can be described by two possible sequences of events (traces) which together can be considered as an event log $L = \{\langle send\ application,\ check\ application,\ notify\ client,\ accept\ application\rangle, \langle create\ application,\ check\ application,\ complete\ application,\ accept\ application\rangle\}$. According to one trace, the client sends a loan application to the bank, then this application is checked. After that, the client is notified, and the application is accepted. The other trace corresponds to a scenario when the application is initially created by a bank employee and checked. After that, the bank employee contacts the client to complete the application. Finally, the application is accepted. Figure 1 presents a workflow net discovered by Inductive miner [4] and Split miner [5] from L. This model accepts two additional traces: $\langle send\ application,\ check\ application,\ complete\ application,\ accept\ application\rangle$, $\langle create\ application, check\ application,\ notify\ client,\ accept\ application\rangle$ not presented in L. These traces violate the business logic of the process. If the application was sent by a client, it is completed, and there is no need to take the $complete\ application$ step. Also, if the application was initially created by a bank employee, the step $complete\ application$ is mandatory.

[1] https://apromore.org

[2] https://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

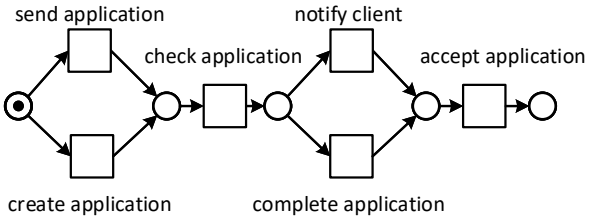


Figure 1: A workflow net discovered from L by Inductive miner and Split miner.

This example demonstrates that the choice between the *notify client* and *complete application* activities depends on the history of the trace. The transition system in Figure 2 shows the behaviour recorded in event log L. State s_1 corresponds to a choice between activities *send application* and *create application*. This choice does not depend on any additional conditions. In contrast, for the system being in states s_4 and s_5, there is no free choice between *notify client* and *complete application* activities. In state s_4, only the *notify client* activity can be executed, while in state s_5, only *complete application* can be performed. This means that there are states in the transition system where the activities *notify client* and *complete application* are not in a free-choice relation (the choice depends on additional conditions and is predefined), while they are in a free-choice relation in the discovered model (Figure 1).

To impose additional restrictions on the process model the state-based region theory can be applied [27, 28, 11]. Figure 2 presents three regions $r_1 = \{s_4, s_5\}$, $r_2 = \{s_2, s_4\}$, and $r_3 = \{s_3, s_5\}$ with outgoing transitions labelled by *notify client* and *complete application* events discovered by the state-based region algorithm [11].

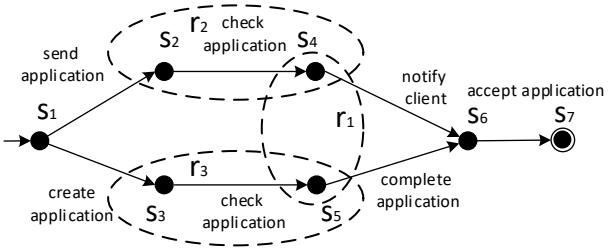


Figure 2: Transition system that encodes event log L.

Figure 3 presents a target workflow net obtained from the initial workflow net (Figure 1) by inserting places that correspond to the discovered regions. As one may note, in addition to r_1, two places r_2 and r_3 were added. These places impose additional constraints, such that the enhanced process model accepts event log L and does not support additional traces and, hence, is more precise.

In the next sections, we describe an approach that implements this idea.

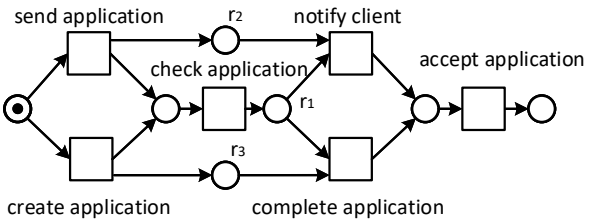


Figure 3: A workflow net enhanced with additional regions (places) r_2 and r_3.

3. Preliminaries

In this section, we formally define event logs and process models, such as transition systems, Petri nets, and workflow nets.

3.1. Sets, Multisets, Event Logs

Let S be a finite set. A *multiset* m over S is a mapping $m : S \to \mathbb{N}_0$, where $\mathbb{N}_0$ is the set of all natural numbers (including zero), i.e., multiset m contains $m(s)$ copies of element $s \in S$.

For two multisets m, m' we write $m \subseteq m'$ iff $\forall s \in S : m(s) \leq m'(s)$ (the inclusion relation). The sum of two multisets m and m' is defined as: $\forall s \in S : (m + m')(s) = m(s) + m'(s)$. The difference of two multisets is a partial function: $\forall s \in S$, such that $m(s) \geq m(s')$, $(m - m')(s) = m(s) - m'(s)$.

Let E be a finite set of events. A *trace* σ (over E) is a finite sequence of events, i.e., $\sigma \in E^*$, where E^* is the set of all finite sequences over E, including the empty sequence of zero length. An *event log* L is a set of traces, i.e., $L \subseteq E^*$.

3.2. Transition Systems, Petri Nets, Workflow Nets

Let S and E be two disjoint non-empty sets of *states* and *events*, $B \subseteq S \times E_\tau \times S$, where $E_\tau = E \cup \{\tau\}$ and $\tau \notin E$ is a special *silent* event, be a *transition relation*. A *transition system* is a tuple $TS = (S, E, B, s_i, S_{fin})$, where $s_i \in S$ is an initial state and $S_{fin} \subseteq S$ – a set of final states. Elements of B are called *transitions*. We write $s \xrightarrow{e} s'$, when $(s, e, s') \in B$ and $s \xrightarrow{e}$, when $\exists s' \in S$, such that $(s, e, s') \in B$; $s \not\xrightarrow{e}$, otherwise. Transition system *TS* is *τ-free* iff $\forall (s, e, s') \in B$ it holds that $e \neq \tau$.

A trace $\sigma = \langle e_1, \ldots, e_m \rangle$ is called *feasible* in *TS* iff $\exists s_1, \ldots, s_n \in S : \; s_i \xrightarrow{\bar{e}_1} s_1 \xrightarrow{\bar{e}_2} \ldots \xrightarrow{\bar{e}_n} s_n$, $n \geq m$, $s_n \in S_{fin}$, and $\langle e_1, \ldots, e_m \rangle = \langle \bar{e}_1, \ldots, \bar{e}_n \rangle |_{\{\tau\}}$, where $\langle \bar{e}_1, \ldots, \bar{e}_n \rangle |_{\{\tau\}}$ is a sequence obtained from $\langle \bar{e}_1, \ldots, \bar{e}_n \rangle$ by removing all τ without changing the order of the remaining elements, i.e., a *feasible* trace leads from the initial state to some final state possibly taking silent transitions. A *language accepted by TS* is defined as the set of all traces feasible in *TS*, and is denoted by $\mathcal{L}(TS)$.

We say that a transition system *TS encodes* an event log L iff it is τ-free and each trace from L is a feasible trace in *TS*, and inversely each feasible trace in *TS* belongs to L. An example of a τ-free transition system is shown in Figure 2. States and transitions are presented by vertices and directed arcs respectively. The initial state s_1 is marked by an additional incoming arrow, while the only final state s_7 is indicated by a circle with double border.

Let P and T be two finite disjoint sets of *places* and *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ be a flow relation. Let also E be a finite set of events, and $l : T \to E_\tau$, where $E_\tau = E \cup \{\tau\}$, $\tau \notin E$, be a labelling function, such that $\forall t_1, t_2 \in T, t_1 \neq t_2$, $l(t_1) \neq \tau$, it holds that $l(t_1) \neq l(t_2)$, i.e., all the *non-silent* transitions are uniquely labelled. Then $N = (P, T, F, l)$ is a *Petri net*. If $\forall t \in T : l(t) \neq \tau$, then N is a *τ-free* Petri net.

A *marking* in a Petri net is a multiset over the set of its places. A marked Petri net (N, m_0) is a Petri net N together with its *initial marking* m_0.

Graphically, places are represented by circles, transitions by boxes, and the flow relation F by directed arcs. Places may carry tokens represented by filled circles. A current marking m is designated by putting $m(p)$ tokens into each place $p \in P$. Marked Petri nets are presented in Figures 1 and 3.

For a transition $t \in T$, an arc (p, t) is called an *input arc*, and an arc (t, p) an *output arc*, $p \in P$. The *preset* ${}^\bullet t$ and the *postset* $t^\bullet$ of transition t are defined as the multisets over P, such that ${}^\bullet t(p) = 1$, if $(p, t) \in F$, otherwise ${}^\bullet t(p) = 0$, and $t^\bullet(p) = 1$ if $(t, p) \in F$, otherwise $t^\bullet(p) = 0$. A transition $t \in T$ is *enabled* in a marking m iff ${}^\bullet t \subseteq m$. An enabled transition t may *fire* yielding a new marking $m' =_{\text{def}} m - {}^\bullet t + t^\bullet$ (denoted $m \xrightarrow{t} m'$, $m \xrightarrow{l(t)} m'$, or just $m \to m'$). We say that m_n is *reachable* from m_1 iff there is a (possibly empty) sequence of firings $m_1 \to \cdots \to m_n$ and denote this relation by $m_1 \xrightarrow{*} m_n$.

$\mathcal{R}(N, m)$ denotes the set of all markings reachable in Petri net N from marking m. A marked Petri net (N, m_0), $N = (P, T, F, l)$ is *safe* iff $\forall p \in P, \forall m \in \mathcal{R}(N, m_0) : m(p) \leq 1$, i.e., at most one token can appear in a place.

A *reachability graph* of a marked Petri net (N, m_0), $N = (P, T, F, l)$, with a labelling function $l : T \to E_\tau$, is a transition system *TS* $= (S, E, B, s_i, S_{fin})$ with the set of states $S = \mathcal{R}(N, m_0)$ and transition relation B defined by $(m, e, m') \in B$ iff $m \xrightarrow{t} m'$, where $e = l(t)$. The initial state in *TS* is the initial marking m_0. If some reachable markings in (N, m_0) are distinguished as final markings, they are defined as final states in *TS*. The *language* of a Petri net (N, m_0), denoted by $\mathcal{L}(N, m_0)$ is the language of its reachability graph, i.e., $\mathcal{L}(N, m_0) = \mathcal{L}(TS)$. We say that a Petri net (N, m_0) *accepts* a trace iff this trace is feasible in the reachability graph of (N, m_0); a Petri net *accepts* a language iff this language is accepted by its reachability graph. When S is finite we can construct a τ-free transition system $\widehat{TS}$, such that $\mathcal{L}(\widehat{TS}) = \mathcal{L}(TS)$ [29], we will call it a *τ-closure* of the reachability graph *TS*.

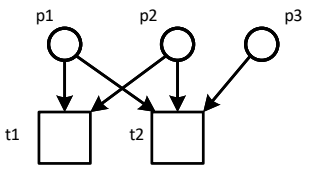


Figure 4: A non-free-choice Petri net.

Given a Petri net $N = (P, T, F, l)$, two transitions $t_1, t_2 \in T$ are in a *free-choice relation* iff ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$ or ${}^\bullet t_1 = {}^\bullet t_2$. When $l(t_1) \neq \tau$ and $l(t_2) \neq \tau$, we also say that events (or activities) $l(t_1)$ and $l(t_2)$ are in a *free-choice relation*. Petri net N is called *free-choice* iff for all $t_1, t_2 \in T$, it holds that t_1 and t_2 are in a free-choice relation. This is one of the several equivalent definitions for free-choice Petri nets presented in [6]. A Petri net is called *non-free-choice* iff it is not free-choice. Figure 4 presents an example of a non-free-choice Petri net, where for two transitions t_1 and t_2 it holds that ${}^\bullet t_1 \cap {}^\bullet t_2 = \{p_1, p_2\} \neq \emptyset$ and ${}^\bullet t_1 = \{p_1, p_2\} \neq {}^\bullet t_2 = \{p_1, p_2, p_3\}$.

The choice of which transition will fire depends on an additional constraint imposed by place p_3.

If $m(p_1) > 0$, $m(p_2) > 0$, and $m(p_3) = 0$, then only t_1 is enabled, thus there is no free-choice between t_1 and t_2. Another example of a non-free-choice Petri net was presented earlier in Figure 3, where transitions labelled by $notify\ client$ and $complete\ application$ are not in a free-choice relation, thus the Petri net is not free-choice. An example of a free-choice Petri net is presented in Figure 1.

Workflow nets are a special subclass of Petri nets designed for modelling workflow processes [30]. A workflow net has one initial and one final place, and every place or transition is on a directed path from the initial to the final place.

Formally, a marked Petri net $N = (P, T, F, l)$ is called a *workflow net* iff

1. There is one source place $i \in P$ and one sink place $o \in P$, such that i has no input arcs and o has no output arcs.
2. Every node from $P \cup T$ is on a directed path from i to o.
3. The initial marking contains one token in the source place.
4. The final markings contain one token in the sink place.

The *language* of a workflow net N is denoted by $\mathcal{L}(N)$.

A workflow net N with the initial marking $[i]$ containing only one token in the source place and the final marking $[o]$ containing only one token in the sink place is *sound* iff

1. For every state m reachable in N, there exists a firing sequence leading from m to the final state $[o]$. Formally, $\forall m : [([i] \xrightarrow{*} m)$ implies $(m \xrightarrow{*} [o])]$;
2. The state $[o]$ is the only state reachable from $[i]$ in N with at least one token in place o. Formally, $\forall m : [([i] \xrightarrow{*} m) \wedge ([o] \subseteq m)$ implies $(m = [o])]$;
3. There are no *dead* transitions in N. Formally, $\forall t \in T\ \exists m, m' : ([i] \xrightarrow{*} m \xrightarrow{t} m')$.

Note that both models presented in Figures 1 and 3 are sound workflow nets.

4. Region State-Based Synthesis

In this section, we give a brief description of the well-known state-based region algorithm [27] applied for the synthesis of Petri nets from transition systems.

Let $TS = (S, E, T, s_i, S_{fin})$ be a τ-free transition system with a finite set of states S and $r \subseteq S$ be a subset of states. Subset r is a *region* iff for each event $e \in E$ one of the following conditions holds:

- all the transitions $s_1 \xrightarrow{e} s_2$ *enter* r, i.e., $s_1 \notin r$ and $s_2 \in r$,
- all the transitions $s_1 \xrightarrow{e} s_2$ *exit* r, i.e., $s_1 \in r$ and $s_2 \notin r$,
- all the transitions $s_1 \xrightarrow{e} s_2$ *do not cross* r, i.e., $s_1, s_2 \in r$ or $s_1, s_2 \notin r$.

In other words, all the transitions labelled by the same event are of the same type (*enter*, *exit*, or *do not cross*) for a particular region.

A region r' is said to be a *subregion* of a region r iff $r' \subseteq r$. A region r is called a *minimal region* iff it does not have any subregions other than r.

The state-based region algorithm covers the transition system by its minimal regions [31]. Figure 5 presents the transition system from Figure 2 covered by minimal regions: $r_1 = \{s_4, s_5\}$, $r_2 = \{s_2, s_4\}$, $r_3 = \{s_3, s_5\}$, $r_4 = \{s_2, s_3\}$, $r_5 = \{s_6\}$, $r_6 = \{s_1\}$, and $r_7 = \{s_7\}$. According

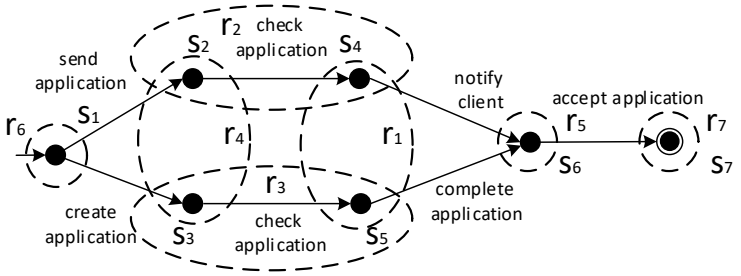


Figure 5: Applying the state-based region algorithm to the transition system presented in Figure 2.

to the algorithm in [27], every minimal region is transformed to a place p in the target Petri net and connected with transitions corresponding to the *exiting* and *entering* events by outgoing and incoming arcs, respectively (refer to Figure 6). If a region contains the initial state of *TS*, the corresponding place p is added to the initial marking of the target Petri net. If a region contains a final state of *TS*, new final markings are obtained by adding p to the existing final markings and these new final markings are added to the overall set of the final markings of the target Petri net.

Region r *separates* two different states $s, s' \in S$, $s \neq s'$, iff $s \in r$ and $s' \notin r$. Finding such a region is the *state separation problem* between s and s' and is denoted by $SSP(s, s')$. When an event e is not enabled in a state s, i.e., $s \stackrel{e}{\not\rightarrow}$, a region r, containing s may be found, such that e does not *exit* r. Finding such a region is known as the *event/state separation problem* between s and e and is denoted by $ESSP(s, e)$.

A well-known result in region theory establishes that if all SSP and $ESSP$ problems are solved, then synthesis is exact [17]:

Theorem 4.1. A τ-free TS with a finite number of states can be synthesized into a safe τ-free Petri net N with the reachability graph isomorphic to TS if all SSP and $ESSP$ problems are solvable.

These problems are also known to be NP-complete [17]. In this paper, we reduce the size of these problems by constructing regions corresponding to particular events only.

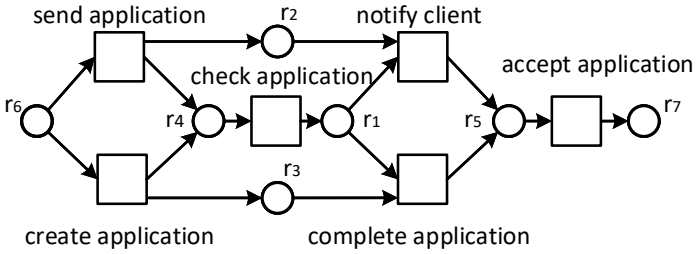


Figure 6: A Petri net synthesized from the transition system presented in Figure 5.

5. Repairing Free-Choice Process Models

In this section, we describe our approach for repairing free-choice workflow nets using non-local constraints captured in the event logs. Additionally, we investigate formal properties of the repaired process models.

5.1. Problem Definition

Let N be a free-choice workflow net discovered from event log L and let *TS* be a transition system encoding L. Due to limitations of the automated discovery methods [4, 5] that construct free-choice workflow nets (even if they discover models from a wider class of workflow nets with silent transitions), not all the places that correspond to minimal regions may have been derived, and therefore important SSP/$ESSP$ problems may not be solved in N, when considering τ-closure of the reachability graph $\widehat{\mathcal{R}}(N, [i])$ as the behaviour to represent with N.

This brings us to the following characterization of the problem. Let $t_1, \ldots, t_n$ be non-silent transitions, i.e., $l(t_1) \neq \tau, \ldots, l(t_n) \neq \tau$, in N with ${}^\bullet t_1 = {}^\bullet t_2 = \cdots = {}^\bullet t_n$, i.e., $t_1, \ldots, t_n$ are in the free-choice relation in N, and let $TS = (S, E, T, s_i, S_{fin})$ be a minimal transition system encoding the event log L. If there exists a state $s \in S$, and $1 \leq i < j \leq n$ such that:

1. e_i, e_j correspond to transitions t_i, t_j, respectively,
2. $s \xrightarrow{e_i}$,
3. $s \not\xrightarrow{e_j}$

Then, the relation of $t_1, \ldots, t_n$ in N corresponds to a *false free-choice relation*, not observed in *TS*.

There is no place in N corresponding to a region that solves the $ESSP(s, e_j)$ problem, because $t_1, \ldots, t_n$ are in a free-choice relation in N. For instance, the Petri net in Figure 1 contains places corresponding to regions r_1, r_4, r_5, r_6, and r_7 shown in Figure 5, and none of those regions solves the $ESSP(s_4, complete\ application)$ and $ESSP(s_5, notify\ client)$ problems in the transition system.

Note that we define the notion of a false free-choice relation for a minimal transition system (transition system with a minimal number of states [29]) encoding the event log. This is done in order

to avoid the case when there exists a state s' which is equivalent to s, such that $s' \xrightarrow{e_j}$. During the minimization, these equivalent states will be merged into one state with outgoing transitions labelled by e_i and e_j showing that there is no false free-choice relation between corresponding transitions. Another reason to minimize the transition system is to reduce the number of states being analysed.

Note that there is no guarantee that an $ESSP$ problem can be solved. Nevertheless, in the running example, regions r_2 and r_3 solve $ESSP(s_4, complete\ application)$ and $ESSP(s_5, notify\ client)$ problems.

5.2. Algorithm Description

In this subsection, we present an algorithm for enhancement of a free-choice workflow net N with additional constraints from event log L (Algorithm 1).

Algorithm 1: RepairFreeChoiceWorkflowNet

Input: Free-choice workflow net N; Event log L.
Output: Repaired net N' obtained from N by inserting additional non-local constraints.

```
/* Construct minimal transition system */
```
1 $TS \leftarrow$ ConstructMinTS(L);

```
/* Compute ESSP problems */
```
2 $ESSPProblems \leftarrow$ FindFalseFreeChoiceRelations(N,TS);

3 $N' \leftarrow N$;

4 **foreach** $(s, e)\ from\ ESSPProblems$ **do**

```
    /* Solve ESSP(s,e) */
```
5 $Y \leftarrow$ ComputeRegionsESSP(TS,s,e);

6 **if** $(Y \neq \emptyset)$ **then**

```
        /* ESSP(s,e) has been solved */
```
7 $N' \leftarrow$ AddNewConstraints(N', Y);

8 **end**

9 **return** N'

Firstly, by applying $ConstructMinTS$, a minimal transitional system encoding the event log L is constructed.[3] Then, false free-choice relations and corresponding $ESSP$ problems are identified. According to the definition of a false free-choice relation presented earlier, the procedure for finding false free-choice relations $FindFalseFreeChoiceRelations$ is polynomial in time. Indeed, to find all the false free-choice relations one needs to check whether all the states of the transition system have none or all outgoing transitions labelled by events assumed to be in free-choice relations within the original workflow net N. When the false free-choice relations are discovered, for each corresponding $ESSP$ problem, the function $ComputeRegionsESSP$, which finds regions solving the $ESSP$ problem, is

[3]Transition system can be constructed from the event log as a prefix-tree [10] with subsequent minimization [29].

applied. Finally, if new regions solving $ESSP$ problems are found, the function *AddNewConstraints* adds the corresponding constraints (places) to the target workflow net N'.

Since the problem of finding minimal regions which solve the $ESSP$ problem is known to be NP-complete, this is the most time consuming step of Algorithm 1.

Suppose that transitions $t_1, \ldots, t_n$ in N labelled by $e_1, \ldots, e_n$, respectively, are in a false free-choice relation. Hence, there exist a state s in *TS* and two events $e_i, e_j, i, j \in [1, \ldots, n]$, such that $s \xrightarrow{e_i}$ and $s \not\xrightarrow{e_j}$. To solve $ESSP(s, e_j)$ problem we need to find a region r, such that $s \in r$ and e_j does not *exit* r (Figure 7a). Then, this region will be converted to a place (Figure 7b) that imposes additional behavioural constraints. Note that region r' in Figure 7a and Figure 7b corresponds to the original free-choice relation.

(a) Solving $ESSP(s, e_j)$ problem.

(b) Adding non-local constraint r.

Figure 7: Finding a new region that solves the $ESSP$ problem.

The goal of the repair algorithm is to find additional constraints for the transitions that are in false free-choice relations. Therefore, we may narrow the search space and consider only those regions containing s where e_i is the exiting event. Since e_i may be involved in several $ESSP$ problems (with different states s and not-exiting transitions e_j), the general approach could be to construct all the minimal regions with the exiting event e_i and check whether these regions do not correspond to the free-choice relation, i.e., not all the events from $\{e_1, \ldots, e_n\}$ are exiting.

According to [32], when constructing regions, the space of potential solutions expands in no more than two directions for each of the events. Suppose that $|E|$ is a set of events, then the time complexity of constructing regions with exiting event $e_i \in E$ can be estimated as $O(2^{|E|-1})$. Suppose that $E' \subseteq E$ is a set of events corresponding to transitions in false free-choice relations, then the overall time complexity of the repair algorithm is $O(|E'| \cdot 2^{|E|-1})$. Although the upper time bound of the algorithm is exponential, the search space is not always expanded in two directions, and often only one search direction is possible, or no search directions are possible at all. Moreover, we do not construct all the regions covering the transition system, but only those that repair the model. In Section 6, we demonstrate that this algorithm can be efficiently applied to repair process models discovered from real-life event data.

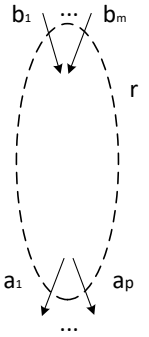
5.3. Formal Properties

In this subsection, we prove the formal properties of Algorithm 1. Firstly, we study the relation between the languages of the initial and target workflow nets. Theorem 5.1 proves that if a trace *fits* the initial model (i.e., the initial model accepts the trace), it also *fits* the target model. Although the proof seems trivial, we consider different cases in order to verify that a final marking is reached.

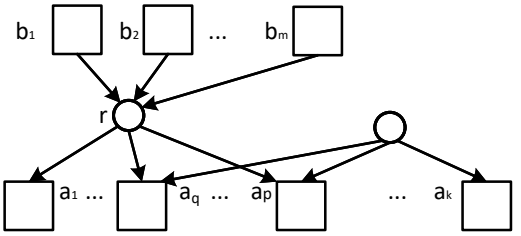
Theorem 5.1. (Fitness)
Let $\sigma \in L$ be a trace of an event log $L \in E^*$, and $N = (P, T, F, l)$, $l : T \to E_\tau$ be a free-choice workflow net, such that its language contains σ, i.e., $\sigma \in \mathcal{L}(N)$. Workflow net $N' = (P \cup P', T, F', l)$, $l : T \to E_\tau$, is obtained from N and L using Algorithm 1. Then the language of N' contains σ, i.e., $\sigma \in \mathcal{L}(N')$.

Proof:
Let us prove that an insertion of a single place by Algorithm 1 preserves the ability of the workflow net to accept trace σ. Consider a place r (Figure 8b) constructed from the corresponding region r (Figure 8a) with entering events $b_1, \ldots, b_m$ and exiting events $a_1, \ldots, a_q, \ldots a_p$. Events $a_q, \ldots, a_p$ can belong to a larger set of events $a_q, \ldots, a_p, \ldots, a_k$ which are in a free-choice relation within N. Consider the workflow net N' with a new place r (the fragment of N' is presented in Figure 8b) and the following four cases:

1. Suppose $\sigma = \langle e_1, \ldots, e_l \rangle \in L$ does not contain events from $\{b_1, \ldots, b_m\}$ and $\{a_1, \ldots, a_p\}$ sets. Since $\sigma \in \mathcal{L}(N)$, there is a sequence of firings in N: $m_0 \xrightarrow{e_1} m_1 \xrightarrow{e_2} \ldots \xrightarrow{e_l} m_n$, where m_0 and m_n are the initial and final markings of the workflow net respectively. The same sequence of firings can be repeated within the target workflow net N', because σ does not contain events from the sets $\{b_1, \ldots, b_m\}$ and $\{a_1, \ldots, a_p\}$, and the place r is not involved in this sequence of firings.



(a) A fragment of transition system encoding L.

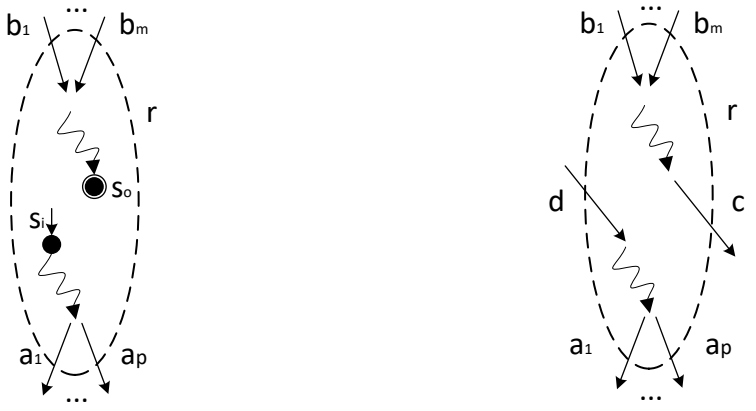


(b) A fragment of N'.

Figure 8: Adding a new place r.

2. Now let us consider trace $\sigma = \langle e_1, \ldots, b_i, \ldots, a_j, \ldots, e_l \rangle$ in which each occurrence of event b_i from the set $\{b_1, \ldots, b_m\}$ is followed by an occurrence of event a_j from $\{a_1, \ldots, a_p\}$. Similarly, for the firing sequence within N: $m_0 \xrightarrow{e_1} m_1 \xrightarrow{e_2} \ldots \xrightarrow{b_i} m_i \rightarrow \ldots \rightarrow m_j \xrightarrow{a_j} \ldots \xrightarrow{e_l} m_n$, there is a corresponding sequence $m_0 \xrightarrow{e_1} m_1 \xrightarrow{e_2} \ldots \xrightarrow{b_i} m'_i \rightarrow \ldots \rightarrow m'_j \xrightarrow{a_j} \ldots \xrightarrow{e_l} m_n$ for N', such that $\forall p \in P : m'_i(p) = m_i(p), m'_i(r) = 1, \forall p \in P : m'_j(p) = m_j(p)$, and $m'_j(r) = 1$.

3. Consider trace σ where an event from $\{b_1, \ldots, b_m\}$ is not followed by an event from the set $\{a_1, \ldots, a_p\}$. More precisely, there are two possible cases: (1) trace σ contains an event from $\{b_1, \ldots, b_m\}$ and does not contain an event from $\{a_1, \ldots, a_p\}$; (2) an occurrence of an event from set $\{b_1, \ldots, b_m\}$ is followed by another occurrence of an event from the same set $\{b_1, \ldots, b_m\}$ and only after that an event from the set $\{a_1, \ldots, a_p\}$ may follow. For the case (1), it is possible that the final state s_o belongs to the region r (Figure 9a). Then, the firing sequence in N: $m_0 \xrightarrow{e_1} m_1 \xrightarrow{e_2} \ldots \xrightarrow{b_i} m_i \rightarrow \ldots \xrightarrow{e_l} m_n$ corresponds to the firing sequence $m_0 \xrightarrow{e_1} m_1 \xrightarrow{e_2} \ldots \xrightarrow{b_i} m'_i \rightarrow \ldots \xrightarrow{e_l} m'_n$ in N', where $\forall p \in P : m'_i(p) = m_i(p), m'_i(r) = 1$, $\ldots, m'_n(p) = m_n(p), m'_n(r) = 1$ and, according to the synthesis algorithm (Section 4), m'_n is a new added final marking.



(a) Start/final state inside the region r.

(b) The bypass incoming/outgoing transitions.

Figure 9: Fragments of the transition system that encodes L.

The other possible scenario for the cases (1) and (2), is that the trace σ does not terminate inside region r. In both cases, there is a transition labelled by an event $c \notin \{a_1, \ldots, a_p\}$ which exits region r (Figure 9b). While it is obvious for the case (1), for the case (2) this can be proven by the fact that there are two occurrences of events from $\{b_1, \ldots, b_m\}$ with no occurrences of events from $\{a_1, \ldots, a_p\}$ in between, and hence the trace σ leaves the region r in order to enter it again with a transition labelled by an event from $\{b_1, \ldots, b_m\}$. Having a new exiting event $c \notin \{a_1, \ldots, a_p\}$ contradicts the definition of the region r which has $\{a_1, \ldots, a_p\}$ as a set of exiting events. Thus, we have proven that there is no such a trace in the initial event log

containing an event from the set $\{b_1, \ldots, b_m\}$ which is not followed by an event from the set $\{a_1, \ldots, a_p\}$.

4. Consider the last possible case when an event from $\{a_1, \ldots, a_p\}$ is not preceded by an event from $\{b_1, \ldots, b_m\}$ in trace σ. Here again we can distinguish two situations: (1) σ contains an event from $\{a_1, \ldots, a_p\}$ and does not contain an event from $\{b_1, \ldots, b_m\}$; (2) the occurrence of an event from $\{a_1, \ldots, a_p\}$ is firstly preceded by another occurrence of an event from $\{a_1, \ldots, a_p\}$ which in its turn can be preceded by an event from $\{b_1, \ldots, b_m\}$. Just like in the previous case, two scenarios are possible: the trace starts inside the region r (Figure 9a.) or there is a transition entering r and labelled by an event $d \notin \{b_1, \ldots, b_m\}$ (Figure 9b). Similarly to the case (3), we can prove that either (1) we adjust the initial marking, or (2) r is not a region.

Thus, we have proved that if a place corresponding to a region constructed by the Algorithm 1 is added to the initial workflow net N then all the traces from L accepted by N are also accepted by the resulting workflow net N'. □

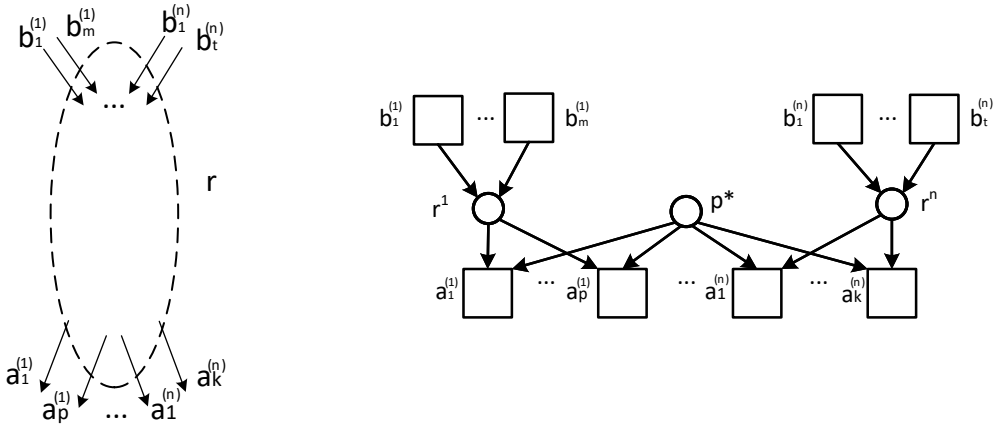
The following theorem states that the resulting model cannot be less precise than the initial process model, i.e., it cannot accept new traces which were not accepted by the initial model.

Theorem 5.2. (Precision)
Let $N = (P, T, F, l)$, $l : T \to E_\tau$, be a free-choice workflow net and let L be an event log over a set of events E. If workflow net N' is obtained from N and L by Algorithm 1, then the language of N contains the language of N', i.e., $\mathcal{L}(N') \subseteq \mathcal{L}(N)$.

Proof:
The proof follows from the well-known result that the addition of new places (preconditions) can only restrict the behaviour and, hence, the language of the Petri net [33]. □

Next, we formulate and prove a sufficient condition for the soundness of the resulting workflow net. This condition is formulated in terms of the state-based region theory.

Theorem 5.3. (Soundness)
Let L be an event log over set E. Let $N = (P, T, F, l)$, $l : T \to E_\tau$ be a sound free-choice workflow net with initial and final markings $[i]$ and $[o]$, respectively. Suppose that workflow net $N' = (P \cup P', T, F', l)$ is obtained from N and L by applying Algorithm 1 to a set of transitions $t_1, \ldots, t_q$ in a free-choice relation within N, such that these transitions are all not silent, i.e., $\forall j \in [1, q] : l(t_j) \neq \tau$. Suppose also that $\{r^{(1)}, \ldots, r^{(n)}\}$ is a set of regions constructed by Algorithm 1 in the transition system encoding L (Figure 10b). Let $E^{(1)}_{ent} = \{b^{(1)}_1, \ldots, b^{(1)}_m\}, \ldots, E^{(n)}_{ent} = \{b^{(n)}_1, \ldots, b^{(n)}_t\}$ and $E^{(1)}_{exit} = \{a^{(1)}_1, \ldots, a^{(1)}_p\}, \ldots, E^{(n)}_{exit} = \{a^{(n)}_1, \ldots, a^{(n)}_k\}$ be sets of entering and exiting events for the regions $r^{(1)}, \ldots, r^{(n)}$ respectively. Consider unions of these sets: $E_{ent} = E^{(1)}_{ent} \cup \ldots \cup E^{(n)}_{ent}$ and $E_{exit} = E^{(1)}_{exit} \cup \ldots \cup E^{(n)}_{exit}$. Suppose that $E_{exit} = \{l(t_1), \ldots, l(t_q)\}$ and $\forall j, k, j \neq k$ holds that $E^{(j)}_{exit} \cap E^{(k)}_{exit} = \emptyset$. If there exists a (not necessarily minimal) region r in the τ-closure of the reachability graph of N (Figure 10a) with entering and exiting sets of events E_{ent} and E_{exit}, respectively, which does not contain states corresponding to $[i]$ (initial) and $[o]$ (final) markings of N, then N' is sound.



(a) A fragment of the reachability graph of N.

(b) A fragment of N'.

Figure 10: Adding new places to N.

Proof:
Repeating the proof of Theorem 5.1 and taking into account that the initial and final states of τ-closure of the reachability graph of N do not belong to the region r, we can state that there is a following relation between E_{ent} and E_{exit} within $\mathcal{L}(N)$, i.e, for each trace, each occurrence of the event from E_{ent} is followed by an occurrence of the event from E_{exit} and there are no other occurrences of events from E_{ent} between them.

The firing sequences of N' which do not involve firings of transitions labelled by events from E_{ent} and E_{exit} repeat the corresponding firing sequences of N and do not violate the soundness of the model.

Let us consider a firing sequence of N' which involves firings of transitions labelled by events from E_{ent} and E_{exit}. Consider $b \in E_{ent}$, the firing sequence enabling and firing b in N': $[i] \xrightarrow{*} m'_1 \xrightarrow{b} m'_2$, corresponds to the firing sequence performed by N: $[i] \xrightarrow{*} m_1 \xrightarrow{b} m_2$, where $\forall p \in P : m_1(p) = m'_1(p)$, $m_2(p) = m'_2(p)$. Without loss of generality, suppose that $b \in E^{(i)}_{ent}$, then $m'_2(r^i) = 1$, where r^i is a place constructed by Algorithm 1.

Since E_{ent} and E_{exit} events are in a following relation within $\mathcal{L}(N)$, they are in the following relation within $\mathcal{L}(N')$, because $\mathcal{L}(N') \subseteq \mathcal{L}(N)$. Consider sequences of steps leading to some of the events from E_{exit}. These firing sequences will be: $m'_2 \xrightarrow{*} m'_3$ and $m_2 \xrightarrow{*} m_3$, where $m_3(p) = m'_3(p)$ and $m'_3(r^i) = 1$, in N' and N respectively.

In model N' transitions labelled by the events from $E^{(i)}_{exit}$ will be enabled in m'_3, all other transitions labelled by events from $E_{exit} \setminus E^{(i)}_{exit}$ have their preceding places empty in m'_3: $m'_3(r^j) = 0$, $i \neq j$.

In workflow net N' it holds that $m'_3(r^i) = 1$ and $m'_3(p^*) = 1$ (p^* is a choice place for the transitions in a free-choice relation within N, see Figure 10b) and hence a step: $m'_3 \xrightarrow{a} m'_4$, where $a \in E^{(i)}_{exit}$ can be performed. After a is fired the place r^i is emptied. A corresponding firing step in N: $m_3 \xrightarrow{a} m_4$ can be taken, because $m_3(p^*) = 1$, and all the transitions labelled by events from E_{exit} are enabled in m_3. These steps lead models to the same markings: $\forall p \in P : m_4(p) = m'_4(p)$ from which firing the same transitions the final marking $[o]$ can be reached. If the rest sequence of firings contains events from E_{ent} and E_{exit}, we repeat the same reasoning.

Thus, we have shown that all the transitions within N' can be fired. Due to the soundness of N, since all the firing sequences of N' correspond to firing sequences of N, and the number of tokens in each place from P in corresponding markings of N' and N coincide, the final marking can be reached from any reachable marking of N' and there are no reachable markings in N' with tokens in the final place o and some other places.

□

5.4. Using High-Level Constructs to Model Discovered Non-Local Constraints

In this subsection, we demonstrate how the discovered process models with non-local constraints can be presented using high-level modelling languages, such as BPMN (Business Process Model and Notation) [3]. Free-choice workflow nets can be modelled by a core set of process modelling elements that includes start and end events, tasks, parallel and choice gateways, and sequence flows. The equivalence of free-choice workflow nets and process models based on the core set of elements is studied in [34, 30]. Most process modelling languages, such as BPMN, support these core elements. A BPMN model corresponding to the discovered free-choice workflow net (shown in Figure 1) is presented in Figure 11.

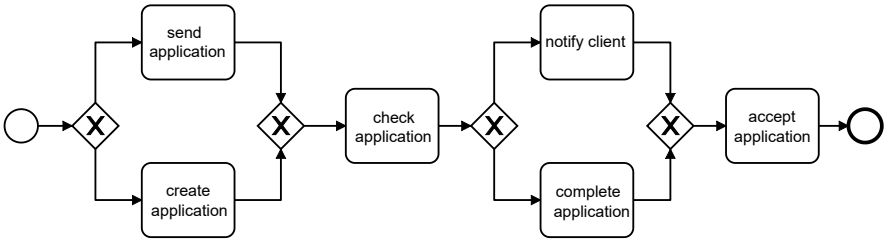


Figure 11: A BPMN model that corresponds to the workflow net in Figure 1.

If a workflow net is not free-choice, it cannot be presented using core elements only [34]. However, the BPMN language offers additional high-level modelling constructs which can be used to model non-free-choice constraints. Figure 12 demonstrates a BPMN model that corresponds to a non-free-choice net (in Figure 3) constructed by Algorithm 1.

In addition to core modelling elements, *signal events* and an *event-based gateway* can be used. Assuming throw signal events are buffered within the scope of the corresponding process instance (this depends on the chosen BPMN engine), the signal events can capture the discovered non-local dependencies. For instance, after the $send\ application$ task is performed, a signal $sent\ by\ client$ is

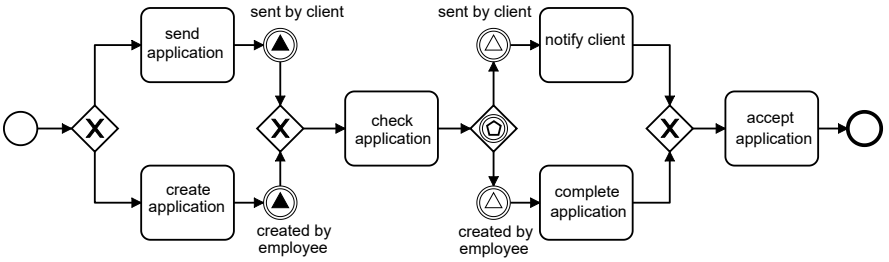


Figure 12: A BPMN model with signals corresponding to the workflow net presented in Figure 3.

thrown. After that, an *event-based gateway* is used to select a branch depending on which of the catching signal events that immediately follow the gateway is fired. For example, if the type of the caught event is signal and its value is $sent\ by\ client$, then task $notify\ client$ is performed.

Besides that, the long-distance dependencies can be modelled in BPMN using data objects and conditional sequence flows [35, 36]. Figure 13 shows the same process where $send\ application$ and $create\ application$ tasks define the value of the $ApplicationType$ data object, and then, depending on its value one of the process branches is activated.

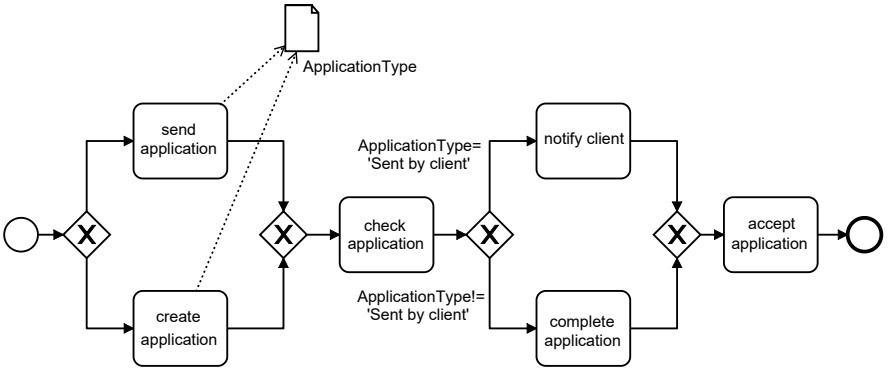


Figure 13: A BPMN model with a data object corresponding to the workflow net presented in Figure 3.

6. Case Study

In this section, we demonstrate the results of applying our approach to synthetic and real-life event logs. The approach is implemented as an Apromore [25] plugin called *"Add long-distance relations"* and is available as part of the Apromore Community Edition. [4] All the results were obtained in real-time using Intel(R) Core(TM) i7-8550U CPU @1.80 GHz with 16 GB RAM.

[4] `https://github.com/apromore/ApromoreCE_ExternalPlugins`

6.1. Synthetic Event Logs

To assess the ability of our approach to automatically repair process models we have built a set of workflow nets with non-local dependencies. An example of one of these workflow nets is presented in Figure 14.

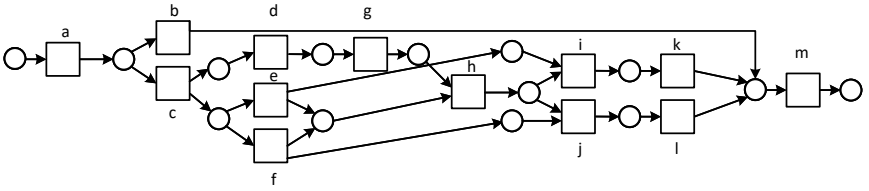


Figure 14: A workflow net used for the synthesis of an event log.

We simulated each of the workflow nets and generated event logs containing accepted traces. After that, from each event log L we discovered a free-choice workflow net N using Split miner. Then, our approach was applied to N and L producing an enhanced workflow net N' with additional constraints. To compare behaviours of N and N' workflow nets, conformance checking techniques [37] assessing fitness (the share of the log behaviour accepted by a model) and precision (the share of the model behaviour captured by the log) were applied. In all the cases, both models N and N' accept all the traces from L showing maximum fitness values of 1.0 (according to Theorem 5.1, if N accepts a trace, then N' also accepts this trace). Precision values as well as the structural characteristic of the workflow nets are presented in Table 1. These results demonstrate that our approach is able to automatically reveal hidden non-local constraints discovering precise workflow nets when applied to synthetic event logs.

Event log	#Transitions / #Places in N	#Transitions / #Places in N'	Precision (N, L)	Precision (N', L)
1	18 / 14	18 / 18	0.972	1.0
2	13 / 12	13 / 14	0.945	1.0
3	10 / 9	10 / 11	0.899	1.0
4	12 / 13	12 / 15	0.911	1.0
5	6 / 4	6 / 6	0.841	1.0

Table 1: Structural (number of transitions and number of places) and behavioural characteristics (precision) of free-choice (N) and enhanced (N') workflow nets.

At the same time, while other approaches for the discovery of non-free-choice workflow nets, such as α++ Miner [8] and the original Petri net synthesis technique [10] can also synthesize precise workflow nets from this set of simple event logs, they often either produce unsound workflow nets with

dead transitions (in the case of α++ Miner), or fail to construct a model in reasonable time (in case of the original synthesis approach) when applied to real-world event logs. In the next subsection, we apply our approach to a real-world event log showing that our approach can discover a more precise and sound process model in a real-world setting.

6.2. Real-life Event Logs

The proposed approach was applied to Business Process Intelligence Challenge (BPIC) real-world event logs, including: (1) the event logs of a university travel expense claims system that processes *Domestic Declarations* (DD-BPIC20) [38], *Prepaid Travel Costs* (PTC-BPIC20) [39], and *Request For Payment* (RP-BPIC20) [40] documents; (2) the *Hospital Billing* (HB-BPIC17) [41] event log obtained from financial modules of a hospital information system; (3) the event log of a *Road Traffic Fine Management* (RTF-BPIC15) [42] system; (4) the *Receipt phase event log of a building permit application process* (RPBP-BPIC14) [43]; (5) the *Detailed Incident Activity* (DIA-BPIC14) [44] event log of an ITIL (Information Technology Infrastructure Library) process that aligns IT services and banking procedures.

To reduce noise and apply the proposed technique to the most frequent behaviour, we filtered each event log, keeping only 20 of its most frequent traces.[5] Then, for each filtered event log L, the Inductive mining algorithm [4] guaranteeing perfect fitness (the language of the discovered model contains all traces from the event log) was applied and a corresponding free-choice workflow net N was discovered. For each pair (N, L), we applied our repair algorithm (Algorithm 1) and constructed a workflow net N' enhanced with additional places. Table 2 presents the characteristics of each log L, such as the overall number of occurrences of events and traces, the number of events, the structural characteristics of workflow nets, such as the size of N (the number of places and transitions) and the number of places added to N', the behavioural characteristics (precision) of N and N' with respect to L, and the overall computation time in milliseconds.

These results demonstrate that although we add extra places, increasing the size of the workflow net, we also improve its precision by imposing additional behavioural constraints. To calculate precision we apply the entropy-based conformance checking metric [37] that is monotonic, i.e., the lower the share of model traces that are not present in the log, the higher the precision. At the same time, we preserve model fitness. According to Theorem 5.1, since the fitness of the initial workflow net is 1.0 (the workflow net accepts all the event log traces), the fitness of the corresponding enhanced workflow net is also 1.0. All the enhanced workflow nets were constructed in less than a second for each of the event logs.

The workflow net N discovered from the real-world event log L of DIA-BPIC14 is presented in Figure 15. The repair algorithm applied to the model N and event log L constructs N' by discovering a *false free-choice relation* for the activities *Quality Indicator Fixed*, *Resolved* and *Update* and adds a place (region) r that also specifies *Caused by CI* (*Caused by Configuration Item)* as an exit event. This new place restricts the model behaviour in such a way that the *Caused by CI* event can occur only if *Resolved* and *Update* events did not occur. According to the synthesis algorithm (Section 4), in addition to the initial m_0 and finial m_f markings of N with one token in places p_0 and p_{12} respectively,

[5] In contrast to the definition of event logs given in Section 3, in real-life event logs, traces can appear multiple times.

N' assumes a final marking m'_f, such that $m'_f(p_{12}) = m'_f(r) = 1$. In contrast to N, N' is not sound because a marking with tokens in places p_{12} and r is reachable. However, according to Theorem 5.1, N' can replay all the traces from L; also, N' is more precise than N, in respect to L, i.e., precision for N is 0.851, while precision for N' is 0.873 (Table 2).

Event log	#Events' occur.	#Traces' occur.	#Unique Events	Size N	#New Places	Prec. (N, L)	Prec. (N', L)	Time (ms)
DD-BPIC20	54,863	10,313	14	72	14	0.270	0.487	133
PTC-BPIC20	14,448	1,727	18	67	26	0.233	0.397	252
RP-BPIC20	35,342	6,694	16	76	14	0.317	0.529	213
HB-BPIC17	401,718	94,056	13	69	3	0.310	0.393	142
RTF-BPIC15	552,379	149,145	11	44	4	0.340	0.434	57
RPBP-BPIC14	7,526	1,328	15	47	6	0.463	0.546	120
DIA-BPIC14	43,828	10,560	10	31	1	0.851	0.873	23

Table 2: Event log characteristics (number of occurrences of events and traces, number of events), size (the number of nodes) of free-choice workflow net N, number of new places in enhanced workflow net N', behavioural characteristics (precision) of N and N' in respect to L, and calculation time in milliseconds.

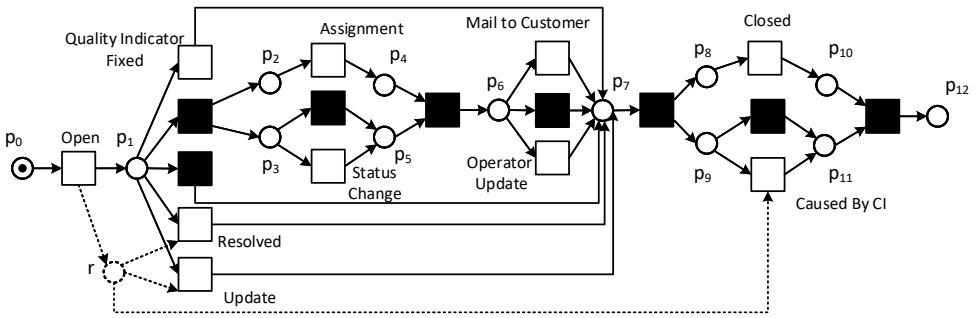


Figure 15: The workflow net N discovered from DIA-BPIC14 event log using the Inductive mining algorithm that guarantees perfect fitness and an enhanced model N' with an additional place r discovered by the repair algorithm. The initial marking m_0 of N' is the marking with only one token in place p_0, i.e., $m_0(p_0) = 1$, for two final markings m_f and m'_f of N', it holds that $m_f(p_{12}) = m'_f(p_{12}) = 1$ and, additionally $m'_f(r) = 1$.

To analyse the calculation times, we compared the proposed repair technique to the region-based approaches that discover process models from event logs without constructing intermediate process representations (Table 3). To that end, we applied the region synthesis algorithm [28] to the transition systems constructed from the real-world event logs described in Table 2. The discovered Petri nets

$\overline{N}$ were obtained in a reasonable time that was higher than the time of applying the proposed repair technique, because not only the regions that solve the outlined $ESSP$ problems were discovered, but also other places. The Petri net models $\overline{N}$ are more compact than the original workflow nets N and N', because they do not contain silent transitions. However, the structure of these Petri nets is not presented by solid control flow graphs and they usually contain unconnected transitions. For instance, the Petri net discovered from the event log DIA-BPIC14 contains five transitions *Quality Indicator Fixed*, *Assignment*, *Status Change*, *Mail to Customer* and *Operator Update* that do not have any incoming or outgoing flows. To discover Petri nets that are represented by a solid control flow, the synthesis algorithm [32] that incorporates the label splitting procedure can be applied. We synthesised workflow nets $\overline{\overline{N}}$ with duplicate transitions (transitions with the identical labels) from the real-world event logs (Table 2) using the ProM (Process Mining Framework) [6] plugin *Convert to Petri net using Regions* [10]. As follows from Table 3, this technique is time consuming and some of the event logs could not be analysed in a reasonable amount of time. The discovered models $\overline{\overline{N}}$ are not block-structured and contain duplicate transitions, however, they are comparable in size to the repaired models N' and clearly represent the control flow structure. Comparing to the other region-based techniques, our repair approach achieves two goals: it constructs structured process models (with additional constraints) in a feasible time.

Event log	Size $\overline{N}$	Disc. time $\overline{N}$ (ms)	Size $\overline{\overline{N}}$	Disc. time $\overline{\overline{N}}$ (ms)
DD-BPIC20	45	669	–	–
PTC-BPIC20	63	2,761	–	–
RP-BPIC20	51	1,056	–	–
HB-BPIC17	37	482	51	630,308
RTF-BPIC15	32	487	43	13,258
RPBP-BPIC14	30	417	62	259,642
DIA-BPIC14	19	235	39	2,462

Table 3: The size (the number of nodes) and the discovery time (in milliseconds) for Petri nets $\overline{N}$ and workflow nets with duplicated labels $\overline{\overline{N}}$, discovered using region-based techniques from the event logs described in Table 2.

7. Conclusion and Future Work

This paper presents an automated repair approach for obtaining precise process models under the presence of non-local dependencies. The approach identifies opportunities for improving the process

[6] `https://www.promtools.org/`.

model by analysing the process behaviour recorded in the input event log. It then uses goal-oriented region-based synthesis to discover new Petri net fragments that introduce non-local dependencies.

The theoretical contributions of this paper have been implemented as an open-source plugin of the Apromore process mining platform. This implementation has then been used to provide experimental results. Based on the experiments conducted, the proposed approach shows good performance.

We foresee different research directions arising from this work. First, implementing the proposed approach for alternative region techniques like language-based [12, 18] or geometric [19, 20] is an interesting avenue to explore. Second, evaluating the impact that well-known problems with event logs, like *noise* or *incompleteness*, may have on the approach, and proposing possible ways to overcome these problems should be explored. Finally, we plan to investigate and classify behavioural characteristics of unsound models discovered by our repair approach.

References

[1] van der Aalst W. Process mining: data science in action. Springer, 2016. ISBN 978-3-662-49850-7.

[2] Carmona J, van Dongen B, Solti A, Weidlich M. Conformance checking - relating processes and models. Springer, 2018. ISBN 978-3-319-99413-0.

[3] OMG. Business Process Model and Notation (BPMN), Version 2.0.2, 2013. URL `http://www.omg.org/spec/BPMN/2.0.2`.

[4] Leemans S, Fahland D, van der Aalst W. Discovering block-structured process models from incomplete event logs. In: ATPN'2014, volume 8489 of *LNCS*, pp. 91–110. Springer, 2014.

[5] Augusto A, Conforti R, Dumas M, La Rosa M, Polyvyanyy A. Split Miner: Automated discovery of accurate and simple business process models from event logs. *Knowl. Inf. Syst.*, 2019. **59**(2):251–284.

[6] Desel J, Esparza J. Free choice Petri nets. Cambridge University Press, USA, 1995. ISBN 0521465192.

[7] Kalenkova A, van der Aalst W, Lomazova I, Rubin V. Process mining using BPMN: Relating event logs and process models process mining using BPMN. *Software and Systems Modeling*, 2017. **16**:1019–1048.

[8] Wen L, Aalst W, Wang J, Sun J. Mining process models with non-free-choice constructs. *Data Min. Knowl. Discov.*, 2007. **15**(2):145–180.

[9] Carmona J, Cortadella J, Kishinevsky M. A region-based algorithm for discovering Petri nets from event logs. In: Business Process Management. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-85758-7, 2008 pp. 358–373.

[10] van der Aalst W, Rubin V, Verbeek H, van Dongen B, Kindler E, Günther C. Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 2010. **9**(1):87.

[11] Solé M, Carmona J. Process mining from a basis of state regions. In: Proceedings of the 31st International Conference on Applications and Theory of Petri Nets, PETRI NETS'10. Springer-Verlag, Berlin, Heidelberg. ISBN 3642136745, 2010 p. 226–245.

[12] van der Werf J, van Dongen B, Hurkens C, Serebrenik A. Process discovery using integer linear programming. In: Applications and Theory of Petri Nets. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-68746-7, 2008 pp. 368–387.

[13] Bergenthum R. Prime miner - process discovery using prime event structures. In: 2019 International Conference on Process Mining (ICPM). 2019 pp. 41–48.

[14] van Zelst S, van Dongen B, van der Aalst W, Verbeek H. Discovering workflow nets using integer linear programming. *Computing*, 2018. **100**(5):529–556.

[15] Mannel L, van der Aalst W. Finding uniwired Petri nets using eST-Miner. In: Business Process Management Workshops. Springer International Publishing, Cham. ISBN 978-3-030-37453-2, 2019 pp. 224–237.

[16] Mannel L, van der Aalst W. Finding complex process-structures by exploiting the token-game. In: 40th International Conference, PETRI NETS'19, Proceedings, volume 11522 of *Lecture Notes in Computer Science*. Springer, 2019 pp. 258–278.

[17] Badouel E, Bernardinello L, Darondeau P. Petri net synthesis. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2015. ISBN 978-3-662-47966-7.

[18] Bergenthum R, Desel J, Lorenz R, Mauser S. Synthesis of Petri nets from finite partial languages. *Fundam. Inform.*, 2008. **88**(4):437–468.

[19] Best E, Devillers R, Schlachter U. A graph-theoretical characterisation of state separation. In: SOFSEM - 43rd International Conference on Current Trends in Theory and Practice of Computer Science, Proceedings. 2017 pp. 163–175.

[20] Schlachter U, Wimmel H. A geometric characterisation of event/state separation. In: Application and Theory of Petri Nets and Concurrency - 39th International Conference, PETRI NETS'18, Proceedings. 2018 pp. 99–116.

[21] Polyvyanyy A, van der Aalst W, ter Hofstede A, Wynn M. Impact-driven process model repair. *ACM Trans. Softw. Eng. Methodol.*, 2016. **25**(4).

[22] Armas-Cervantes A, van Beest N, La Rosa M, Dumas M, García-Bañuelos L. Interactive and incremental business process model repair. In: On the Move to Meaningful Internet Systems. OTM 2017 Conferences. Springer International Publishing, Cham. ISBN 978-3-319-69462-7, 2017 pp. 53–74.

[23] Fahland D, van der Aalst W. Model repair — aligning process models to reality. *Information Systems*, 2015. **47**:220 – 243.

[24] Mitsyuk A, Lomazova I, Shugurov I, van der Aalst W. Process model repair by detecting unfitting fragments. In: AIST 2017, CEUR Workshop Proceedings. 2017 pp. 301–313.

[25] La Rosa M, Reijers H, van Der Aalst W, Dijkman R, Mendling J, Dumas M, García-Bañuelos L. APROMORE: An advanced process model repository. *Expert Systems with Applications*, 2011. **38**(6):7029–7040.

[26] Kalenkova A, Carmona J, Polyvyanyy A, La Rosa M. Automated repair of process models using non-local constraints. In: 41st International Conference, PETRI NETS'20, Proceedings, volume 12152 of *Lecture Notes in Computer Science*. Springer, 2020 pp. 280–300.

[27] Cortadella J, Kishinevsky M, Lavagno L, Yakovlev A. Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers*, 1998. **47**(8):859–882.

[28] Carmona J, Cortadella J, Kishinevsky M. New region-based algorithms for deriving bounded Petri nets. *IEEE Trans. Computers*, 2010. **59**(3):371–384.

[29] Hopcroft J, Ullman J. An n log n algorithm for detecting reducible graphs. In: Proe. 6th Annual Princeton Conf. on Inf. Sciences and Systems. 1972 pp. 119–122.

[30] van der Aalst W, Hirnschall A, Verbeek H. An alternative way to analyze workflow graphs. In: Advanced Information Systems Engineering. Springer Berlin Heidelberg. ISBN 978-3-540-47961-1, 2002 pp. 535–552.

[31] Desel J, Reisig W. The synthesis problem of Petri nets. *Acta Inf.*, 1996. **33**(4):297–315.

[32] Cortadella J, Kishinevsky M, Lavagno L, Yakovlev A. Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers*, 1998. **47**(8):859–882.

[33] Reisig W. Petri nets: An introduction. Springer-Verlag, Berlin, Heidelberg, 1985. ISBN 0387137238.

[34] Favre C, Fahland D, Völzer H. The relationship between workflow graphs and free-choice workflow nets. *Information Systems*, 2015. **47**:197 – 219.

[35] Meyer A, Pufahl L, Fahland D, Weske M. Modeling and enacting complex data dependencies in business processes. In: Business Process Management. Springer Berlin Heidelberg. ISBN 978-3-642-40176-3, 2013 pp. 171–186.

[36] Kalenkova A, Burattin A, de Leoni M, van der Aalst W, Sperduti A. Discovering high-level BPMN process models from event data. *Business Process Management Journal*, 2019. **25**(5):995–1019.

[37] Polyvyanyy A, Solti A, Weidlich M, Di Ciccio C, Mendling J. Monotone precision and recall measures for comparing executions and specifications of dynamic systems. *ACM Trans. Softw. Eng. Methodol.*, 2020. **29**(3).

[38] van Dongen B. BPI challenge 2020: Domestic declarations, 2020. doi:10.4121/uuid:3f422315-ed9d-4882-891f-e180b5b4feb5. URL `https://data.4tu.nl/articles/dataset/BPI_Challenge_2020_Domestic_Declarations/12692543/1`.

[39] van Dongen B. BPI challenge 2020: Prepaid travel costs, 2020. doi:10.4121/uuid:5d2fe5e1-f91f-4a3b-ad9b-9e4126870165. URL `https://data.4tu.nl/articles/dataset/BPI_Challenge_2020_Prepaid_Travel_Costs/12696722/1`.

[40] van Dongen B. BPI challenge 2020: Request for payment, 2020. doi:10.4121/uuid:895b26fb-6f25-46eb-9e48-0dca26fcd030. URL `https://data.4tu.nl/articles/dataset/BPI_Challenge_2020_Request_For_Payment/12706886/1`.

[41] Mannhardt F. Hospital billing - event log, 2017. doi:10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741. URL `https://data.4tu.nl/articles/dataset/Hospital_Billing_-_Event_Log/12705113/1`.

[42] de Leoni M, Mannhardt F. Road traffic fine management process, 2015. doi:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5. URL `https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249/1`.

[43] Buijs J. Receipt phase of an environmental permit application process ('WABO'), CoSeLoG project, 2014. doi:10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6. URL `https://data.4tu.nl/articles/dataset/Receipt_phase_of_an_environmental_permit_application_process_WABO_CoSeLoG_project/12709127/1`.

[44] van Dongen B. BPI challenge 2014: Activity log for incidents, 2014. doi:10.4121/uuid:86977bac-f874-49cf-8337-80f26bf5d2ef. URL `https://data.4tu.nl/articles/dataset/BPI_Challenge_2014_Activity_log_for_incidents/12706424/1`.