Computing and Informatics, Vol. 40, 2021, 446–468, doi: 10.31577/cai_2021_2_446

METHOD FOR REPAIRING PROCESS MODELS WITH SELECTION STRUCTURES BASED ON TOKEN REPLAY

Erjing Bai, Na Su

Qingdao Huanghai University
Qingdao 266427, China

Yu Liang

College of Electronics and Information Engineering
Tongji University
Shanghai 201804, China

Liang Qi*, Yuyue Du

College of Computer Science and Engineering
Shandong University of Science and Technology
Qingdao 266590, China
e-mail: 1832678460@qq.com, yydu001@163.com

Abstract. Enterprise information systems (EIS) play an important role in business process management. Process mining techniques that can mine a large number of event logs generated in EIS become a very hot topic. There always exist some deviations between a process model of EIS and event logs. Therefore, a process model needs to be repaired. For the process model with selection structures, the mining accuracy of the existing methods is reduced because of the additional self-loops and invisible transitions. In this paper, a method for repairing Logical-Petri-nets-based process models with selection structures is proposed. According to the relationship between the input and output places of a sub-model, the de-

* Corresponding author

viation position is determined by a token replay method. Then, some algorithms are designed to repair the process models based on logical Petri nets. Finally, the effectiveness of the proposed method is illustrated by some experiments, and the proposed method has relatively high fitness and precision compared with its peers.

Keywords: Logic Petri net, model repair, token replay, choice structures, process model

1 INTRODUCTION

Business process management has significantly promoted the development of company business processes with the help of advanced enterprise information systems (EIS). Meanwhile, a large number of event logs are generated every day [1]. These event logs can be mined which in turn improve EIS [2] and further the competitiveness of enterprises or organizations. Process mining can extract valuable information from the event logs and improve the actual process models [3]. Process mining techniques mainly include process discovery, conformance checking, and process enhancement [1, 2, 3, 4]. For process discovery, a process model can be built by mining the existing event logs. Conformance checking can find the deviations between a process model and the event logs [4]. For process enhancement, a process model can be expanded and improved by further studying event logs [5]. At present, many process discovery algorithms have been proposed by scholars. A reasonable workflow model with complete logs can be mined based on α algorithm [6], but non-free-choice structures and invisible transitions cannot be well handled. Some extension methods of α algorithm are proposed in [7, 8] to deal with the above problems. A proposed approach based on the genetic algorithm [9, 10] can guarantee a certain quality standard, but it restricts the accurate discovery of block-structured process models and has high computational complexity. A repairing method proposed by Fahland et al. has high fitness [5], but the precision is low because of self-loops and invisible transitions. A single activity with self-loops is inserted into the original models based on Goldratt's and Knapsack's method [11], but the precision is still not high.

A process model is described by Petri nets in [12], since Petri nets have the advantages of rigorous mathematical definitions and powerful graphic display. The static and dynamic states of business processes can be described in Petri nets. However, the existing process models cannot completely replay event logs when business processes or environments are changed. The repaired can replay most of the logs without breaking the main structure of the original model [13]. In this paper, a method for repairing process models with selection structures based on logical Petri nets is proposed. First, model deviations are determined by calculating missing and remaining-tokens; then, the process model is repaired according to the deviations.

The proposed method has higher fitness and precision than Fahland's and Goldratt's methods [5, 11].

The rest of the paper is organized as follows. Section 2 presents some preliminaries and briefly reviews some important concepts. Section 3 presents an approach to repair models with selection structures based on a token replay method via logical Petri nets. The results and performance analysis of simulation experiments are given in Section 4. Section 5 concludes this paper and discusses the future work.

2 PRELIMINARIES

Some basic concepts are introduced in this section including multi-sets [3], tuple [14], event logs, projection, Petri nets [14], logical Petri nets [15], process trees, and workflow nets.

Definition 1 (Multi-sets [3])**.** S is a set. A multi-set Z over S is denoted by $Z : S \to N^+$, N^+ represents a set of positive integers. $\beta(S)$ represents all multi-sets over S.

Definition 2 (Tuple [14])**.** A tuple consisting of n elements is denoted by $x = (a_1, a_2, \ldots, a_n) \in S \times \cdots \times S$, where S is a set. $\pi_i(x)$ is the i^{th} element of x, where $i \in (1, 2, \ldots, n)$.

Definition 3 (Trace and event log [16])**.** Let A be a set of actives. $\sigma \in A^*$ is a trace if $1 \leq i < j \leq \sigma : \sigma[i] \neq \sigma[j]$, and it is a queue of actives. $(\&\sigma)$ represents the set of all activities in trace σ. An event log is a finite nonempty multi-set of trace σ, denoted as $L \in \beta(A^*)$.

For example, given an activity set $A = \{t_1, t_2, t_3, t_4\}$, $\sigma = \langle t_2 t_3 t_1 t_4 \rangle$ is a trace and $(\&\sigma) = \{t_2, t_3, t_1, t_4\}$.

Definition 4 (Projection)**.** Let β be a multi-set over A, $Q \subseteq A$, and $\sigma \in A^*$. $\sigma|Q$ denotes the projection of σ on Q, and $\beta|Q$ denotes the projection of β on Q.

For example, if $\sigma = \langle aabc \rangle$, $Q = \{a, c\}$, and $\beta = [a^3, b, c^2]$, then $\sigma|Q = \langle aac \rangle$ and $\beta|Q = [a^3, c^2]$.

Definition 5 (Petri net)**.** A Petri net is a four-tuple $PN = (P, T; F, M)$, where P is a finite place set, T is a finite transition set, and $F \subseteq (P \times T) \cup (T \times P)$ is a finite arc set, where

1. $N = (P, T; F)$ is a net;
2. $M : P \to \{0, 1, 2, \ldots\}$ is a marking of N; and
3. The transition firing rules of Petri nets are as follows:
   (a) For transition $t \in T$, if $\forall p \in^\bullet t : M(p) \geq 1$, then t is enabled at M, denoted as $M[t >$.

(b) If $M[t>$, then transition t can fire at M, and it generates a new marking M', denoted as $M[t > M'$. For $\forall p \in P$, we have

$$M'(P) = \begin{cases} M(P) - 1, & p \in {}^\bullet t - t^\bullet; \\ M(P) + 1, & p \in t^\bullet - {}^\bullet t; \\ M(P), & \text{otherwise.} \end{cases}$$

Definition 6 (Pre-set and post-set [15]). Let $N = (P, T; F)$ be a net. For $x \in P \cup T$, ${}^\bullet x$ is the pre-set of x if ${}^\bullet x = \{y | y \in P \cup T \wedge (y, x) \in F\}$. $x^\bullet$ is the post-set of x if $x^\bullet = \{y | y \in P \cup T \wedge (x, y) \in F\}$.

Definition 7 (Workflow net [17]). $WFN = (P, T; F, M, i, o)$ is a workflow net, where P, T, F and M can constitute a Petri net; i represents an input place and o represents an output place, where

1. There is an input place $i \in P$, ${}^\bullet i = \phi$ and M_i is an initial marking;
2. There is an output place $o \in P$, $o^\bullet = \phi$, and M_o is a final marking; and
3. $x \in P \cup T$ is always on the path from i to o.

Definition 8 (Logic Petri net). A logic Petri net is a six-tuple denoted by $LPN = (P, T; F, I, O, M)$, where

1. P represents a finite set of places;
2. $T = T_D \cup T_I \cup T_O$ represents a finite set of transitions, and $T \cap P = \phi$. If $t \in T_I \cap T_O$, then ${}^\bullet t \cap t^\bullet = \phi$. T_D represents a set of traditional transitions in Petri nets. T_I represents a set of logic input transitions. For $\forall t \in T_I$, ${}^\bullet t$ is restricted by a logical input expression $f_I(t)$. T_O represents a logic output transitions set. For $\forall t \in T_O$, $t^\bullet$ is restricted by a logical output expression $f_O(t)$;
3. $F = (P \times T)(T \times P)$ is a finite set of arcs;
4. I is a mapping from logic input transitions to logic input functions, and for $\forall t \in T_I$, $I(t) = f_I(t)$;
5. O is a mapping from logic output transitions to logic output functions, and for $\forall t \in T_O$, $O(t) = f_O(t)$;
6. $M : P \to \{0, 1, 2, \dots\}$ is the marking function; and
7. The transition firing rules are as follows:

   (a) For $\forall t \in T_D$, the transition firing rules are the same as in Petri net;
   (b) For $\forall t \in T_I$, if $f_I(t)|M = {}_\bullet T_\bullet$, then a logic input transition can be fired, denoted as $M[t > M'$, and for $\forall p \in {}^\bullet t$, $M'(p) = 0$; and for $\forall p \notin {}^\bullet t \cup t^\bullet$, $M'(p) = M(p)$; and for $\forall p \in t^\bullet$, $M'(p) = 1$; and
   (c) For $\forall t \in T_O$, if $\forall p \in {}^\bullet t$, $M(p) = 1$, then a logic output transition can be fired, and for $\forall p \in {}^\bullet t$, $M'(p) = 0$; for $\forall p \in t^\bullet$ must satisfies $f_O(t)|M = {}_\bullet T_\bullet$, and for $\forall p \notin {}^\bullet t \cup t^\bullet : M'(p) = M(p)$.

For example, a logic Petri net is shown in Figure 1. t_1 is an input transition. $I(t_1) = p_1 \vee p_2$ is the logic input function of t_1. From firing t_1, there are three situations:

1. p_1 contains a token, or
2. p_2 contains a token, or
3. both p_1 and p_2 contain a token.

t_3 is an output transition. $O(t_3) = (p_6 \otimes p_7) \wedge p_8$ is the logic output function of t_3. There are two situations when t_3 is fired: each of p_6 and p_8 contain a token, or each of p_7 and p_8 contain a token.

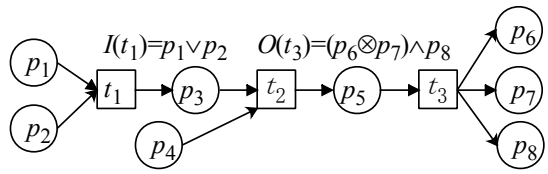


Figure 1. A logic Petri net model LPN_1

Definition 9 (Process tree [18])**.** Let A be a set of actives. $\oplus$ is a given operator set, and τ is an invisible transition, where

1. $a \in A \cup \{\tau\}$ is a process tree;
2. If $PT_1, \ldots, PT_n$ $(n > 0)$ are process trees, then $\oplus(PT_1, \ldots, PT_n)$ is also a process tree; and
3. There are 4 operators:
   - $\times$ stands for a selection relation, and only one sub-tree can occur among $PT_1, \ldots, PT_n$;
   - $\rightarrow$ represents a sequence relation, and the corresponding sub-trees will occur in sequence;
   - $\wedge$ denotes a parallel relation, and the corresponding sub-trees will occur simultaneously; and
   - $\circlearrowleft$ represents a loop structure, and PT_1 denotes a circulatory body, and PT_2, $\ldots$, PT_n $(n \geq 2)$ denotes a loop path.

3 MODEL REPAIRING OF SELECTION STRUCTURES

When business processes or actual working environment changes, event logs generated by actual processes cannot be completely replayed by its original model. Therefore, deviations between an actual event log and its original model should be

identified, and the original process model can be repaired accordingly. In this section, a repairing method is proposed based on a token replay method for models with selection structures. It can find deviations between the original model and the generated logs. Therefore, an original model can be repaired.

3.1 Model Repairing with the Equal Number of Transitions and Log Activities

When a model is repaired, it is necessary to determine the location of deviations between a model and logs. By replaying event log L in a model, the missing and remaining-tokens can be calculated [19]. The position of deviations can be determined. In token replay, tokens will dynamically change from the initial place to the final place when a trace is completely consistent with a model. If there are deviations between a trace and a model, tokens cannot reach the end place according to the missing-tokens, and the fitness of the rest trace cannot be analyzed. Thus an enhanced replaying algorithm is given next.

Algorithm 1 Enhanced Replaying Algorithm

Input: A Workflow net $WFN = (P, T; F, i, o)$ and an event log $L \in \beta(\sigma^*)$;
Output: M.

1: **for** each $p \in P$ **do**
2: **if** $p = i$ **then**
3: $M(i) = 1$;
4: **else**
5: $M(p) = 0$;
6: **end if**
7: **end for**
8: **for** $(j = 1; t_j \in \&(\sigma); j++)$ **do**
9: **if** $t_j \in T$ and $p \in {}^\bullet t_j - t_j^\bullet$ **then**
10: $M(p) \leftarrow M(p) - 1$;
11: **end if**
12: **if** $t_j \in T$ and $p \in t_j^\bullet - {}^\bullet t_j$ **then**
13: $M(p) \leftarrow M(p) + 1$;
14: **end if**
15: **end for**
16: $M(o) \leftarrow M(o) - 1$;
17: **return** M.

In Algorithm 1, all places are initialized in Steps 1–7. There is a token in the initial place, and there is no token in other places. The transitions in traces are replayed in Steps 8–15. If $t_j \in T$ and $p \in {}^\bullet t_j - t_j^\bullet$, then $M(p) = M(p) - 1$. If $t_j \in T$ and $p \in t_j^\bullet - {}^\bullet t_j$, then $M(p) = M(p) + 1$. In Step 16, a token is consumed from the output place, and a final marking is obtained in Step 17. The computational complexity of Algorithm 1 is $O(n)$.

Example 1. A workflow net WFN_1 is shown in Figure 2 where $\sigma_1 = \langle t_1, t_2, t_4, t_5, t_6, t_7 \rangle$ is replayed. Table 1 shows the change of tokens based on Algorithm 1.

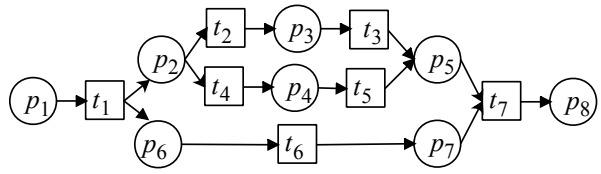


Figure 2. Workflow net model WFN_1

$M(p_8) = 0$ at the end of a replay. After executing Algorithm 1, $M(p_2) = -1$ represents that a token is missing, and $M(p_3) = 1$ represents that a token is remaining. For missing-token places, there should be an arc connecting a place such that tokens can be generated. For remaining-token places, there should be an arc connecting a transition to consume a token. The locations of missing and remaining-token places represent an end position and a start position of an adding arc, respectively. When a model is repaired, another side of the connecting arc needs to be calculated. For example, ${}^\bullet p_3$ should connect t_4 in Figure 2. To repair WFN_1, the arc from p_3 to t_4 should be added based on $\sigma_1 = \langle t_1, t_2, t_4, t_5, t_6, t_7 \rangle$. Another side of an added arc can be determined based on the start position p_2 or the end position p_5 in selection structures. From a process tree and selection relation pairs, start and end pairs are defined with selection structures.

Transisions	$M(p_1)$	$M(p_2)$	$M(p_3)$	$M(p_4)$	$M(p_5)$	$M(p_6)$	$M(p_7)$	$M(p_8)$
Start	1	0	0	0	0	0	0	0
t_1	0	1	0	0	0	1	0	0
t_2	0	0	1	0	0	1	0	0
t_4	0	−1	1	1	0	1	0	0
t_5	0	−1	1	0	1	1	0	0
t_6	0	−1	1	0	1	0	1	0
t_7	0	−1	1	0	0	0	0	1
End	0	−1	1	0	0	0	0	0

Table 1. Change of token in WFN_1

Definition 10 (Selection relation)**.** Let PT be a process tree of $WFN = (P, T; F, M, i, o)$. $n =$ "$\times$" is a node of PT, and it represents a selection structure. $C_{RP} = (t_1, t_2)$ is called a selection relation where $t_1 = n_l$ and $t_2 = n_r$ where n_l and n_r represent the leftmost and rightmost subtree of a selection structure, respectively.

S_{CRP} represents a set of selection relations, i.e., $S_{CRP} = \{(t_1, t_2) \mid t_1 = n_l, t_2 = n_r, \forall n = \times\}$. For example, $S_{CRP} = \{(t_2, t_5)\}$ in Figure 3.

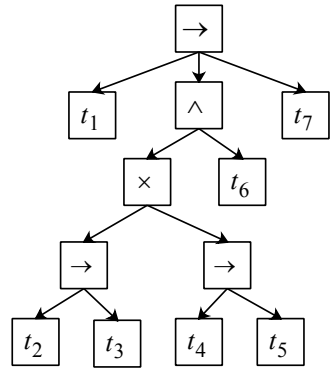


Figure 3. The process tree PT_1 of WFN_1

Definition 11 (Selection structure)**.** PT is a process tree of $WFN = (P, T; F, M, i, o)$. $n =$ "×" is a node of PT, and it represents a selection structure. $C_{RP} \in S_{CRP}$ represents a selection relation pair. $C_{SEP} = (p_1, p_2)$ represents the start and end pair of a selection structure, and $p_1 = {}^\bullet(\pi_1(C_{RP}))$, $p_2 = (\pi_2(C_{RP}))^\bullet$. S_{CSEP} represents a start and end pair set of selection structures. $S_{CSEP} = \{(p_1, p_2) \mid p_1 = {}^\bullet(\pi_1(C_{RP})), p_2 = (\pi_2(C_{RP}))^\bullet, \forall C_{RP} \in S_{CRP}\}$.

For example, $C_{RP} = (t_2, t_5)$, and ${}^\bullet t_2 = \{p_2\}$, $t_5^\bullet = \{p_5\}$, in Figure 3. Therefore, $C_{SEP} = (p_2, p_5)$, and $S_{CSEP} = \{(p_2, p_5)\}$. Thus, an algorithm of calculating S_{CSEP} is given as follows.

Algorithm 2 gives a calculating method of start and end pair sets with selection structures. S_{CRP} and S_{CSEP} are initialized in Step 1. Steps 2–15 find out the leftmost and rightmost sub-tree pairs of all selection structures in WFN, and they are saved in S_{CRP}. Steps 16–18 calculate the start and end pairs of selection structures, according to S_{CRP}, and store them in S_{CSEP}. Step 19 returns S_{CSEP}.

Definition 12 (Mapping function)**.** Given log activity $a \in \&(\sigma)$ and a transition t corresponding to it, $Map(a, t)$ is the mapping function from a log activity to a model transition for trace σ in log L.

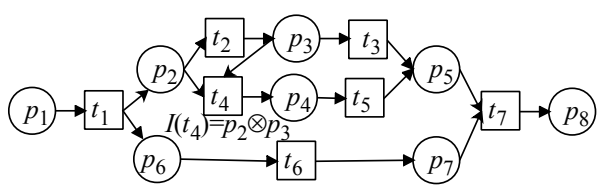


Figure 4. A repaired model of WFN_1 with L_1

Algorithm 3 gives a model repairing method. Step 1 calls Algorithm 1 to replay the logs. Step 2 calls Algorithm 2 to calculate the start and end pair set of

Algorithm 2 S_{CSEP} Calculation

Input: A Workflow net WFN, the non-leaf node of process tree PT denoted by n;
Output: A start and end pair set of selection structures, S_{CSEP}.

1: $S_{CRP} \leftarrow \phi$, $S_{CSEP} \leftarrow \phi$;
2: **for** each $n \in PT$ **do**
3: **if** $n! = \phi$ and $n \in \oplus$ **then**
4: **if** $n =$ "$\times$" **then**
5: $S_{CRP} \leftarrow S_{CRP} \cup \{(n_l, n_r)\}$;
6: **else**
7: **for** all the sub nodes $SN \in n$ **do**
8: $n \leftarrow SN$;
9: Skip to Setp 3;
10: **end for**
11: **end if**
12: **else**
13: break;
14: **end if**
15: **end for**
16: **for** $C_{RP} \in S_{CRP}$ **do**
17: $S_{CSEP} \leftarrow S_{CSEP} \cup \{({}^{\bullet}(\pi_1(C_{RP})), (\pi_2(C_{RP}))^{\bullet})\}$;
18: **end for**
19: **return** S_{CSEP}.

S_{CSEP}. Steps 3–13 calculate a pre-set for remaining-places, remaining-places, the number of remaining-places, a post-set for missing-places, missing-places, and the number of missing-places. Their results are stored in $T_s[\ ]$, $P_s[\ ]$, m, $T_q[\ ]$, $P_q[\ ]$, and n, respectively. All remaining-places and missing-places are judged whether they belong to a start or end place of selection structures in Steps 14–29. If the answer is no, an arc should be added from a remaining-place to $T_q[i]$, and a logic input expression $I(T_q[i]) \leftarrow P_s[j] \otimes P_q[i]$ should be added, for the remaining-places. Moreover, an arc should be added from $T_s[j]$ to $P_q[i]$, and a logic output expression $O(T_s[j]) \leftarrow P_s[j] \otimes P_q[i]$ should be added too, for the missing-places. Steps 30 and 31 mean P' and T' are the same as P and T, respectively. Finally, Algorithm 3 returns a repaired model LPN in Step 32. The computational complexity of Algorithm 3 is $O(n^3)$.

Example 2. Algorithm 3 is used to repair WFN_1 in Figure 2 where $L_1 = \{\langle t_1, t_2, t_4, t_5, t_6, t_7\rangle\}$. The repaired result is shown in Figure 4.

Example 3. Algorithm 3 is used to repair WFN_1 in Figure 2 where $L_2 = \{\langle t_1, t_2, t_3, t_5, t_6, t_7\rangle\}$. The repaired result is shown in Figure 5.

Algorithm 3 Model Repairing Algorithm

Input: A workflow net $WFN = (P, T; F, M, i, o)$ and an event log $L \in \beta(\sigma^*)$;
Output: A repaired logic Petri net, denoted by $LPN = (P, T; F, I, O, M)$.

Call Algorithm 1 to replay event logs;
Call Algorithm 2 to calculate the set of a start and end pair S_{CSEP};
for $(i = 1;\ i < |T|;\ i++)$ **do**
    $m = n = 0$;
    if $M(p_i) > 0$ **then**
        $T_s[m] \leftarrow {}^\bullet p_i$;
        $P_s[m] \leftarrow p_i$;
        $m++$;
    end if
    if $M(p_i) < 0$ **then**
        $T_q[n] \leftarrow p_i^\bullet$;
        $P_q[n] \leftarrow p_i$;
        $n++$;
    end if
end for
for $(j = 1;\ j <= m;\ j++)$ **do**
    for $(i = 1;\ i <= n;\ i++)$ **do**
        for each $\sigma \in L(k = 1;\ k < |\sigma|;\ k++)$ **do**
            if $T_s[j] \in \sigma_k$ and $T_q[i] \in \sigma_{k+1}$ **then**
                if $P_s[j] \notin \pi_1(S_{CSEP})$ and $P_s[j] \notin \pi_2(S_{CSEP})$ **then**
                    $F' \leftarrow F' \cup P_s[j] \rightarrow T_q[i]$;
                    $I(T_q[i]) \leftarrow P_s[j] \otimes P_q[i]$;
                end if
                if $P_q[i] \notin \pi_1(S_{CSEP})$ and $P_q[i] \notin \pi_2(S_{CSEP})$ **then**
                    $F' \leftarrow F' \cup T_s[j] \rightarrow P_q[i]$;
                    $O(T_s[j]) \leftarrow P_s[j] \otimes P_q[i]$;
                end if
            end if
        end for
    end for
end for
$P' \leftarrow P$;
$T' \leftarrow T$;
return $LPN = (P', T'; F', I, O, M)$.

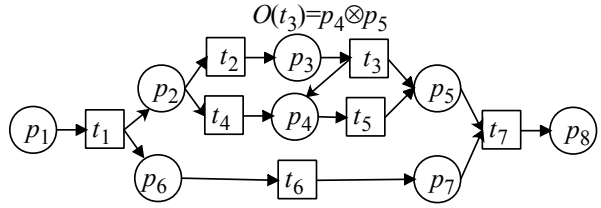


Figure 5. A repaired model of WFN_1 with L_2

3.2 Model Repairing with Different Number of Transitions and Log Activities

It is supposed that model activities and log activities are the same in Algorithm 3. However, log activities generated by an actual process are generally more than model activities. When this situation happened, Algorithm 3 cannot complete a model repairing. To take with this problem, newly added log activities need to be calculated first. A sub-model can be mined according to the newly added log activities and the inductive algorithms. Finally, a sub-model can be inserted into the original model. The algorithm for calculating the newly added log activities is given as follows.

Algorithm 4 New Log Activities

Input: A workflow net $WFN = (P, T; F, M, i, o)$ and an event log $L \in \beta(\sigma^*)$;
Output: The set of newly added log activities, denoted by $NewAct$.

1: $NewAct \leftarrow \phi$, $T_M \leftarrow \phi$;
2: **for** $(i = 1; i <= |T|; i++)$ **do**
3: $\quad T_M \leftarrow T_M \cup \{t_i\}$;
4: **end for**
5: **for** each $\sigma \in L$ **do**
6: $\quad$ **for** $(j = 1, a_j \in \sigma; j <= |\sigma|; j++)$ **do**
7: $\quad\quad Map(a_j, t_k)$;
8: $\quad\quad$ **if** $t_k \notin T_M$ **then**
9: $\quad\quad\quad NewAct \leftarrow NewAct \cup \{a_j\}$;
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: **end for**
13: **return** $NewAct$.

New log activities are calculated in Algorithm 4. Step 1 initializes $NewAct$ and T_M as an empty set. From Steps 2–4, all model activities are added in T_M. Log activities are mapped to model activities in Steps 5–13, and it is judged whether the model activities belong to T_M. If the answer is no, then the log activities are added to $NewAct$. Step 14 returns $NewAct$ of new activities. The computational complexity of Algorithm 4 is $O(n^2)$.

WFN_2 is shown in Figure 6, and $L_3 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\} = \{\langle t_1, t_2, t_4, t_8\rangle, \langle t_1, t_9, t_5, t_6, t_{12}, t_8\rangle, \langle t_1, t_3, t_5, t_6, t_7, t_8\rangle, \langle t_1, t_9, t_{10}, t_{12}, t_8\rangle, \langle t_1, t_9, t_{11}, t_{12}, t_8\rangle\}$. Newly added log activities t_9, t_{10}, t_{11} and t_{12} can be calculated based on Algorithm 4.

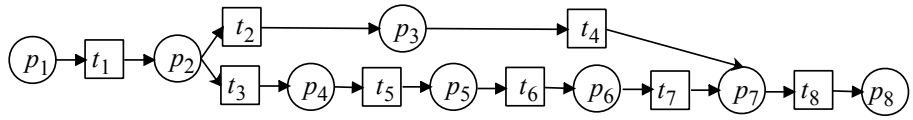


Figure 6. Workflow net model WFN_2

Algorithm 5 Order Relation from an Event Log

Input: An event log $L \in \beta(\sigma^*)$;
Output: An event log relation set R.

1: $R \leftarrow \phi$, $R' \leftarrow \phi$;
2: **for** each $\sigma \in L$ **do**
3: $R' \leftarrow R' \cup \{a_i >_L a_{i+1}\}$;
4: **if** $a \in \sigma$ and $b \notin \sigma$ or $b \in \sigma$ and $a \notin \sigma$ **then**
5: $R \leftarrow R \cup \{a \#_L b\}$;
6: **end if**
7: **if** $a >_L b$ and $b \not>_L a$ **then**
8: $R \leftarrow R \cup \{a \rightarrow_L b\}$;
9: **end if**
10: **if** $a >_L b$ and $b >_L a$ **then**
11: $R \leftarrow R \cup \{a||_L b\}$;
12: **end if**
13: **end for**
14: **return** R.

Algorithm 5, the order relationship of logs can be found. $a \rightarrow_L b$, and $a||_L b$, $a\#_L b$ represent a causality, parallel, choice relation between a and b, respectively. The computational complexity of this algorithm is $O(n)$.

A repairing method of including sub-models with selection structures is shown in Algorithm 6. Algorithm 4 is called to calculate a new log activities set *NewAct* in Step 1. Step 2 calculates the projection of *NewAct* in L to find sub log *SL*. *InductiveMiner*(*SL*) algorithm is called to mine a sub-model $WFN' = (P', T'; F', M', i', o')$ in Step 3. Step 4 calls Algorithm 5 to calculate the order relation of event logs. A sub-model is inserted into the original model in Steps 5–13. Step 14 calls Algorithm 3 to repair the model.

Example 4. Workflow net WFN_2 can be repaired based on Algorithm 6, and $L_3 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\} = \{\langle t_1, t_2, t_4, t_8\rangle, \langle t_1, t_9, t_5, t_6, t_{12}, t_8\rangle, \langle t_1, t_3, t_5, t_6, t_7, t_8\rangle, \langle t_1, t_9, t_{10}, t_{12}, t_8\rangle, \langle t_1, t_9, t_{11}, t_{12}, t_8\rangle\}$.

Algorithm 6 Model Repairing Method for Sub-Models with Choice Structures

Input: A workflow net $WFN = (P, T; F, M, i, o)$ and an event log $L \in \beta(\sigma^*)$;
Output: The repaired logic Petri net $LPN = (P', T'; F', I, O, M)$.

1: Call Algorithm 4 to calculate a new log activities set $NewAct$;
2: Calculating the projection of $NewAct$ in L to find its sub log SL;
3: Call $InductiveMiner(SL)$ algorithm to mine a sub-model $WFN' = (P', T'; F', i', o', M')$;
4: Call Algorithm 5 to calculate the order relationship of event logs;
5: **for** each $(t_i \rightarrow_L i'^{\bullet}) \in R$ **do**
6: $\quad F' = F' \cup (t_i^{\bullet} \rightarrow i'^{\bullet})$;
7: **end for**
8: **for** each $({}^{\bullet}o' \rightarrow_L t_i) \in R$ **do**
9: $\quad F' = F' \cup ({}^{\bullet}o' \rightarrow {}^{\bullet}t_i)$;
10: **end for**
11: $F' = F \cup F' - (i' \rightarrow i'^{\bullet}) - ({}^{\bullet}o' \rightarrow o')$;
12: $T = T \cup T'$;
13: $P = P \cup P' - i' - o'$;
14: Call Algorithm 3 to repair the model.

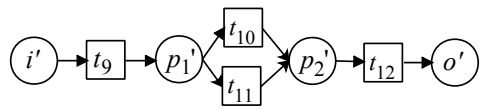


Figure 7. Sub Workflow net model WFN'

$NewAct = \{t_9, t_{10}, t_{11}, t_{12}\}$ is calculated first according to the Algorithm 4. Then the sub log $SL = \{\langle t_9, t_{12}\rangle, \langle t_9, t_{10}, t_{12}\rangle, \langle t_9, t_{11}, t_{12}\rangle\}$ is calculated based on $NewAct$ and L_2. A sub-model WFN' can be mined according to the inductive algorithm, and the result shows in Figure 7. The relationship between the new log activities t_9–t_{12} and the original model transition is calculated. Besides, the relationships are $t_1 \rightarrow t_9$, $t_2 \# t_9$, $t_3 \# t_9$, and $t_{12} \rightarrow t_8$. Finally, WFN' is inserted into the original

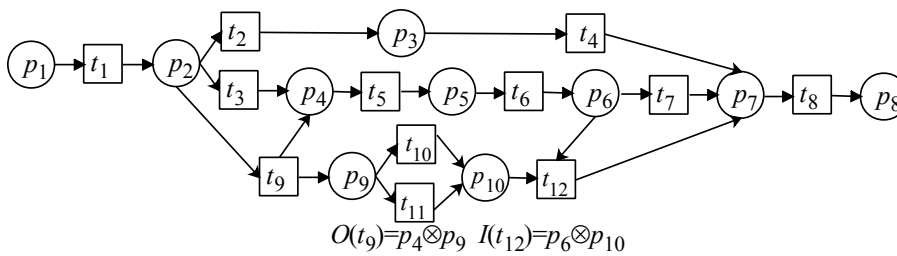


Figure 8. Repaired model of WFN_2

model and the model is repaired according to Algorithm 3. The repaired model is shown in Figure 8.

4 SIMULATION EXPERIMENTS

Simulation experiments are conducted in this section. The data is from an inspection department from a hospital in Qingdao, China, and event logs can be accessible at: `https://pan.baidu.com/s/1AG4TvrxF62uAP2QOowOSEQ`. The experimental results of the proposed method in this paper are compared and analyzed with those of Fahland's method [5] and Goldratt's method [11], where the former is implemented by the corresponding plug-ins in ProM 6.10 available at `http://www.promtools.org/`; and the latter is implemented in the DOS window and edited in ProM 6.10.

4.1 Model Repairing

The Petri net model in Figure 9 can be mined by α algorithm [7] according to event logs, and it shows the whole process of patients from outpatient appointments to treatment and departure. First, a patient makes an appointment at the triage station or by phone call. Then he (or she) needs to book and then gets a reservation number. Some patients may not make an appointment, but they need to register at first. After that, the patients need to wait for calling their number and are inquired by the doctor. Then the patients may need to do some examinations, i.e., common CT, PET-CT, chest enhanced scan, ESR, biochemical full set, blood gas analysis, and blood routine. Then, the doctor makes a diagnosis according to the results of examinations and decides whether the patients leave the hospital or be hospitalized.

The event logs L_1–L_3 are shown in Table 2 including the number and length of traces, and the number of events and activities.

Logs	Traces	Events	Activities	Length
L_1	105	980	24	$7 \sim 11$
L_2	210	1 960	24	$7 \sim 11$
L_3	314	2 938	24	$7 \sim 11$

Table 2. Event logs

There are some deviations between event logs and the process model in Figure 9 since there are some new occurring activities. For example, a patient can make an appointment by internet or WeChat. Besides, the doctor may choose other decisions after diagnosis, i.e., referral or conservative treatment.

The model in Figure 9 can be repaired by three methods. The repaired model of Fahland's method is shown in Figure 10. The repaired model of Goldratt's method is shown in Figure 11. The repaired model of our approach is shown in Figure 12. From Figures 10, 11 and 12 the repaired models of Fahland's and Goldratt's methods

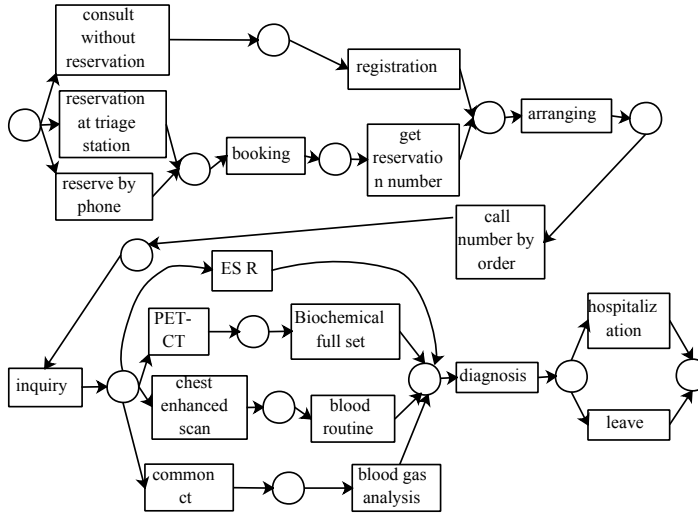


Figure 9. A Petri net of thoracic surgery

add some self-loops and invisible transitions to improve the fitness of models. Self-loops and invisible transitions can decrease the model precision, and increase the model complexity.

The following situation will occur according to actual occurrence logs in this paper. After the common CT, it is possible to check blood gas analysis, ESR, blood routine, biochemical full set, or entering the diagnosis directly. In this paper, suppose that the logic output function of t_{12} is $O(t_{12}) = p_9 \otimes p_{10} \otimes p_{11} \otimes p_{12}$. After checking chest enhanced scan, blood routine, biochemical full set, or directly entering the diagnosis can be done. Thus the logic output function of t_{13} is set as $O(t_{13}) = p_{10} \otimes p_{11} \otimes p_{12}$. After checking PET-CT, biochemical full set or directly entering the diagnosis can be done. The logic output function of t_{14} is set as $O(t_{14}) = p_{11} \otimes p_{12}$. Common CT can be done before checking ESR. Therefore, the logic input function of t_{16} is set as $I(t_{16}) = p_8 \otimes p_9$. After inquiry blood routine, biochemical full set, blood gas analysis or directly entering the diagnosis can be done. Therefore, the logic input function of t_{15}, t_{17}, t_{18}, t_{19} are set as $I(t_{15}) = p_8 \otimes p_9$, $I(t_{17}) = p_8 \otimes p_{10}$, $I(t_{18}) = p_8 \otimes p_{11}$, $I(t_{19}) = p_8 \otimes p_{12}$, respectively.

In the process model of Figure 12, the mapping relationship between transitions and activities is shown in Table 3.

4.2 Model Evaluation

The simplicity of the repaired process model can be analyzed by three repairing methods according to the principle of Occam's Razor [20]. The increased number of places, transitions, invisible transitions and arcs are compared in Table 4, after

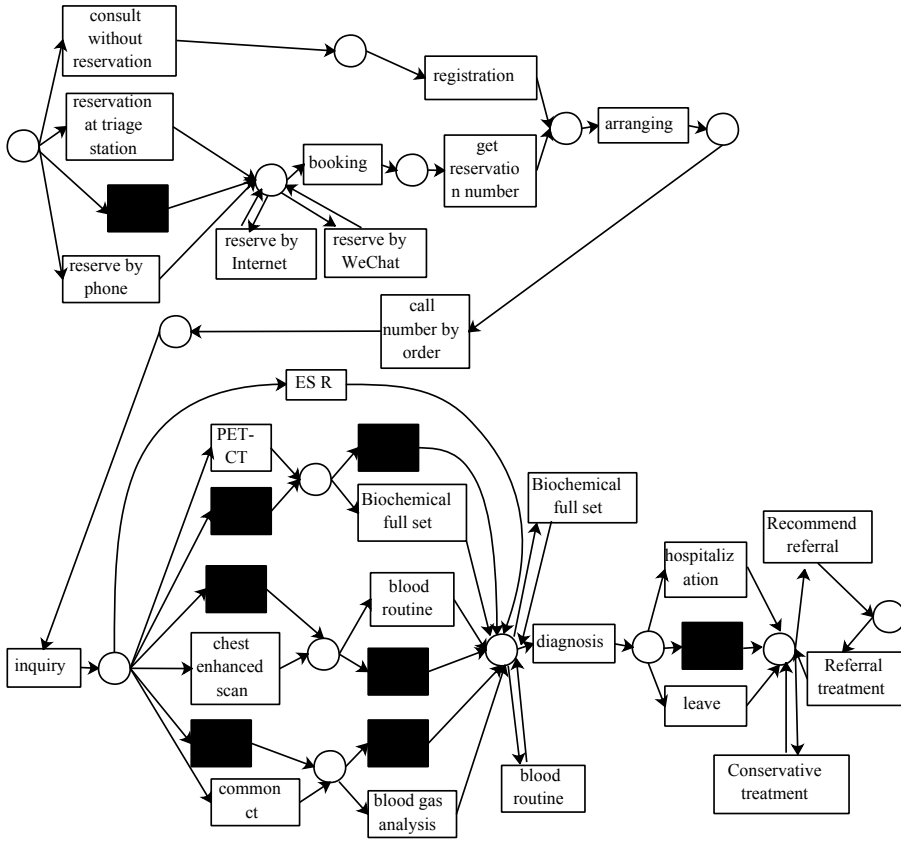


Figure 10. Fahland's method

Transitions	Activities	Transitions	Activities
t_1	reserve by phone	t_{13}	chest enhanced scan
t_2	reservation at triage station	t_{14}	PET-CT
t_3	reserve by Internet	t_{15}	blood gas analysis
t_4	reserve by WeChat	t_{16}	ESR
t_5	consult without reservation	t_{17}	blood routine
t_6	booking	t_{18}	biochemical full set
t_7	get reservation number	t_{19}	diagnosis
t_8	registration	t_{20}	leave
t_9	arranging	t_{21}	hospitalization
t_{10}	call number by order	t_{22}	conservative treatment
t_{11}	inquiry	t_{23}	recommend referral
t_{12}	common CT	t_{24}	referral treatment

Table 3. The mapping relationship of transitions and activities in Figure 9

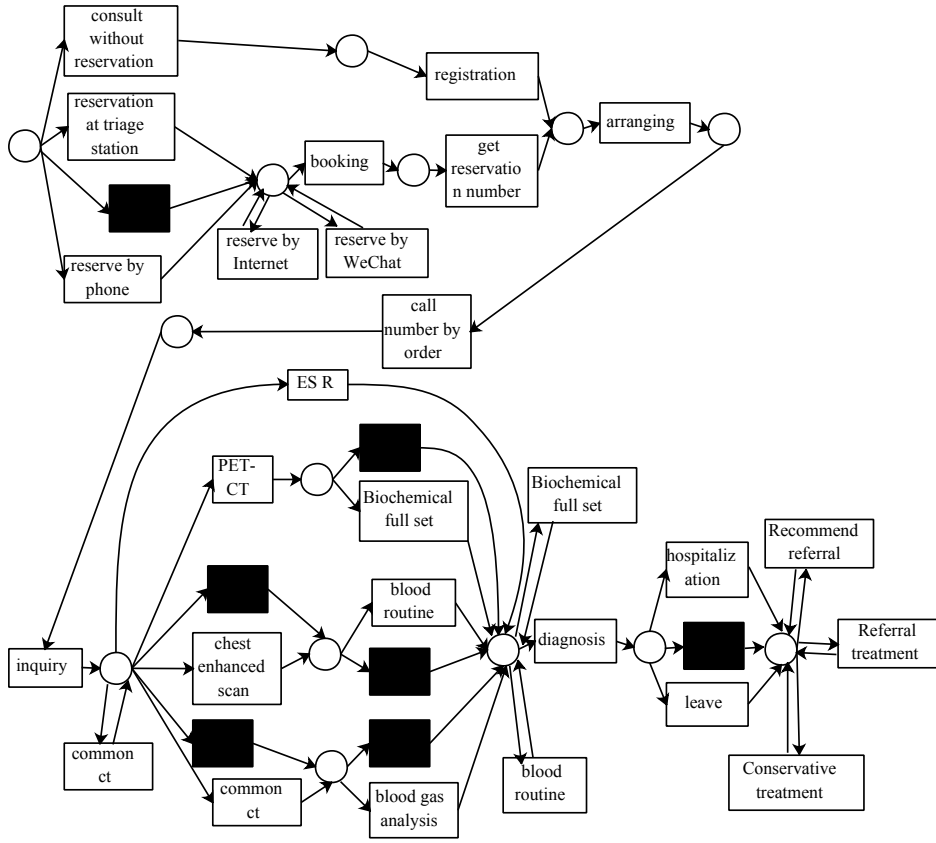


Figure 11. Goldratt's method

the model is repaired by three methods. From the analysis, the invisible transitions are not added in our method, while 8 and 7 invisible transitions are increased by Fahland's and Goldratt's methods, respectively. From the increasing number of arcs, 21 arcs are added by the repairing method of this paper, while 30 arcs are increased by Fahland's and Goldratt's methods, respectively. Thus, the simplicity of our approach is lower than the two repairing methods.

Models	Places	Transitions	Invisible transitions	Arcs
Our approach	1	5	0	21
Fahland's method	1	5	8	30
Goldratt's method	0	5	7	30

Table 4. Comparison of simplicity in three repairing models

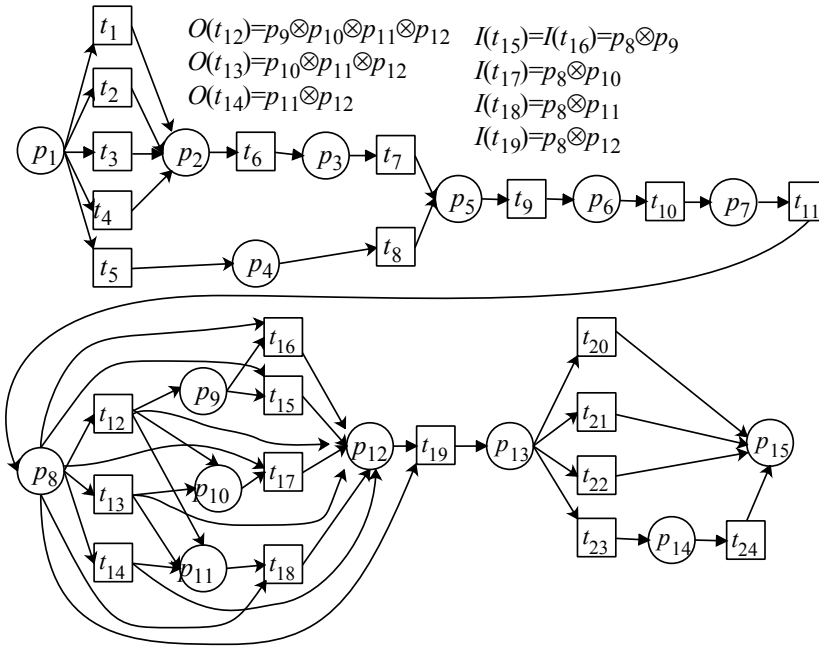


Figure 12. The repaired model by our approach

The fitness [21] is another important metric of conformance checking of process models. The fitness of different models is analyzed based on the number of traces in Figure 13. The fitness of Fahland's and Goldratt's methods are calculated according to the tool of ProM 6.10 plug-in "Replay a Log on Petri Net for Performance Analysis". The fitness of the logical Petri net model proposed in this paper is calculated manually according to reference [21]. From Figure 13, the fitness of every method is more than 0.9. besides, Fahland's method is slightly lower than our approach and Goldratt's method. The fitness of our approach is 1.

The precision of the three repairing methods is shown in Figure 14. The precision of Fahland's and Goldratt's methods are calculated by plug-in "Check Precision based on Align-ETCformance" in ProM 6.10. The precision of the logical Petri net model proposed in this paper is calculated manually according to reference [21]. When the given amount of available repair resources is small, the repaired model by Goldratt's method cannot display all the new log activities. If the value of R is set to 16 in Goldratt's method, all log activities can appear in the repaired model. Therefore, the precision of Goldratt's method is relatively low, around 0.67. The precision of Fahland's method is about 0.73. The precision of our approach is about 0.85. Obviously, our approach has higher precision than others.

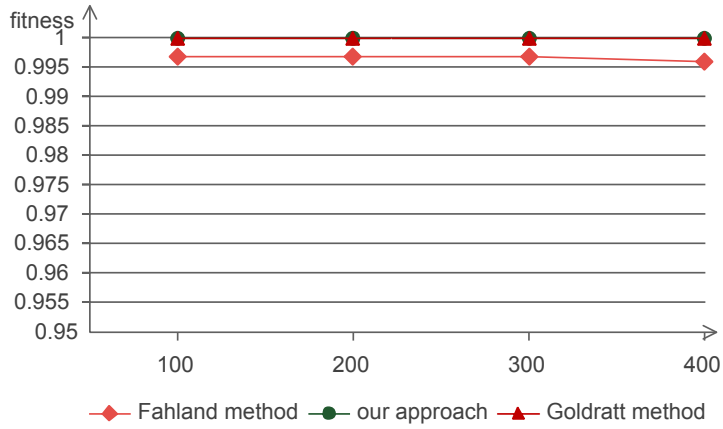


Figure 13. The fitness between different models

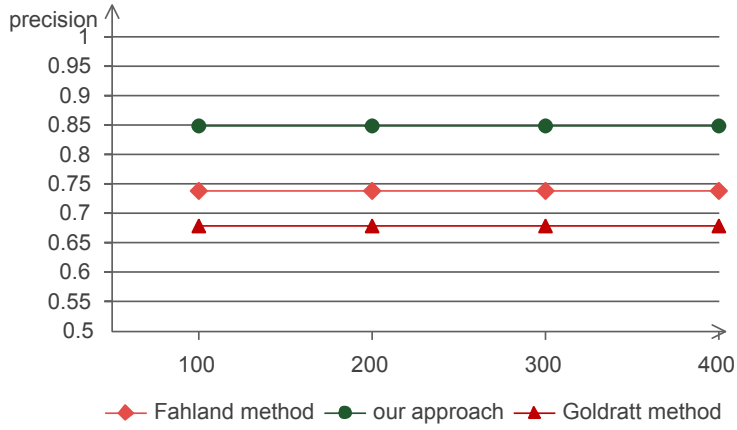


Figure 14. The precision between different models

5 CONCLUSIONS

Aiming at the low precision of Fahland's and other repairing methods, a repairing method of token replay is proposed based on logical Petri nets in this paper. The deviation locations are firstly determined by token replay. A sub-model of newly added log activities is mined by the inductive algorithm. Then an insertion location of the sub-model is determined based on the relationship between the input and output places of the sub-model and event logs. Finally, an original model can be repaired by given algorithms via logical Petri nets. The repairing methods of process models with selection structures are discussed in this paper. Therefore, the

repairing methods of process models with parallel structures or loop structures will be analyzed in our future research.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61903229 and Grant No. 61973180, and in part by the China Electronics Technology Group Corporation (CETC).

REFERENCES

[1] van der Aalst, W. M. P.—Stahl, C.: Modeling Business Processes: A Petri Net Oriented Approach. The MIT Press, Cambridge, USA, 2011, doi: 10.7551/mitpress/8811.001.0001.

[2] Conforti, R.—Dumas, M.—García-Bañuelos, L.—La Rosa, M.: BPMN Miner: Automated Discovery of BPMN Process Models with Hierarchical Structure. Information Systems, Vol. 56, 2016, pp. 284–303, doi: 10.1016/j.is.2015.07.004.

[3] Wang, L.—Du, Y. Y.—Liu, W.: Aligning Observed and Modelled Behaviour Based on Workflow Decomposition. Enterprise Information Systems, Vol. 11, 2017, No. 8, pp. 1207–1227, doi: 10.1080/17517575.2016.1193633.

[4] Wang, Y. Y.—Du, Y. Y.: Comformance Checking Based on Extended Footprint Matrix. Journal of Shandong University of Science and Technology (Natural Science), Vol. 37, 2018, No. 2, pp. 9–15.

[5] Fahland, D.—Van der Aalst, W. M. P.: Model Repair – Aligning Process Models to Reality. Information Systems, Vol. 47, 2015, pp. 220–243, doi: 10.1016/j.is.2013.12.007.

[6] van der Aalst, W. M. P.—Weijters, T.—Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering, Vol. 16, 2004, No. 9, pp. 1128–1142, doi: 10.1109/tkde.2004.47.

[7] Wen, L. J.—van der Aalst, W. M. P.—Wang, J. M.—Sun, J. G.: Mining Process Models with Non-Free-Choice Constructs. Data Mining and Knowledge Discovery, Vol. 15, 2007, No. 2, pp. 145–180, doi: 10.1007/s10618-007-0065-y.

[8] Wen, L. J.—Wang, J. M.—Sun, J. G.: Mining Invisible Tasks from Event Logs. In: Dong, G., Lin, X., Wang, W., Yang, Y., Yu, J. X. (Eds.): Advances in Data and Web Management (APWeb 2007, WAIM 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4505, 2007, pp. 358–365, doi: 10.1007/978-3-540-72524-4_38.

[9] van der Aalst, W. M. P—de Medeiros, A. K. A.—Weijters, A. J. M. M.: Genetic Process Mining. In: Ciardo, G., Darondeau, P. (Eds.): Application and Theory of Petri Nets 2005 (ICATPN 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3536, 2005, pp. 48–69, doi: 10.1007/11494744_5.

[10] de Medeiros, A. K. A.—Weijters, A. J. M. M.—van der Aalst, W. M. P.: Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery, Vol. 14, 2007, No. 2, pp. 245–304, doi: 10.1007/s10618-006-0061-7.

[11] Polyvyanyy, A.—van der Aalst, W. M. P.—Ter Hofstede, A. H. M.—Wynn, M. T.: Impact-Driven Process Model Repair. ACM Transactions on Software Engineering and Methodology, Vol. 25, 2017, No. 4, Art. No. 28, 60 pp., doi: 10.1145/2980764.

[12] Du, Y. Y.—Qi, L.—Zhou, M. C.: A Vector Matching Method for Analyzing Logic Petri Nets. Enterprise Information Systems, Vol. 5, 2011, No. 4, pp. 449–468, doi: 10.1080/17517575.2010.541943.

[13] Rozinat, A.—van der Aalst, W. M. P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems, Vol. 33, 2008, No. 1, pp. 64–95, doi: 10.1016/j.is.2007.07.001.

[14] Teng, Y. X.—Qi, L.—Du, Y. Y.: A Logic Petri Net-Based Repair Method of Process Models with Incomplete Choice and Concurrent Structures. Computing and Informatics, Vol. 39, 2020, No. 1-2, pp. 264–297, doi: 10.31577/cai_2020_1-2_264.

[15] Wang, Z.—Du, Y. Y.—Qi, L.: Extended Colored Logic Petri Net and Its Reachability Analysis. Journal of Shandong University of Science and Technology (Natural Science), Vol. 39, 2020, No. 1, pp. 84–98.

[16] Qi, H. D.—Du, Y. Y.—Liu, W.: Process Model Repairing Method Based on Reachable Markings. Journal of Shandong University of Science and Technology (Natural Science), 2017, pp. 118–124.

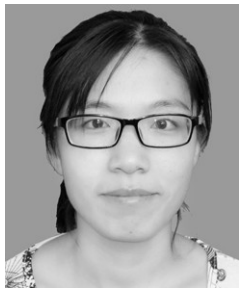
[17] Wen, L.—Wang, J.—van der Aalst, W. M. P.—Huang, B.—Sun, J.: Mining Process Models with Prime Invisible Tasks. Data and Knowledge Engineering, Vol. 69, 2010, No. 10, pp. 999–1021, doi: 10.1016/j.datak.2010.06.001.

[18] Buijs, J. C. A. M.—van Dongen, B. F.—van der Aalst, W. M. P.: A Genetic Algorithm for Discovering Process Trees. Proceedings of the 2012 IEEE Congress on Evolutionary Computation (CEC), 2012, pp. 1–8, doi: 10.1109/cec.2012.6256458.

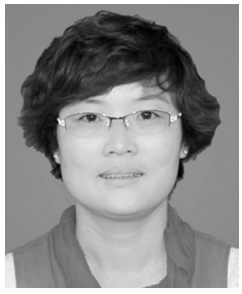
[19] Adriansyah, A.—van Dongen, B. F.—van der Aalst, W. M. P.: Conformance Checking Using Cost-Based Fitness Analysis. Proceedings of the 2011 IEEE 15th International Enterprise Distributed Object Computing Conference (EDOC), 2011, pp. 55–64, doi: 10.1109/edoc.2011.12.

[20] Witten, I.—Frank, E.: Data Minning: Practical Machine Learning Tools and Techniques. Second Edition. Morgan Kaufmann, 2005.

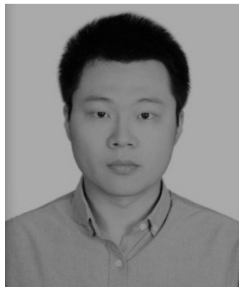
[21] Adriansyah, A.: Aligning Observed and Modeled Behavior. Ph.D. Thesis, Technische Universiteit Eindhoven, 2014, pp. 139–149, doi: 10.6100/IR770080.



Erjing Bai received her B.Sc. degree from the Hebei Normal University, Hebei, China, in 2000, her M.Sc. degree from the Qingdao University of Science and Technology, Qingdao, China, in 2014. She is currently Associate Professor at the College of Qingdao Huanghai University, Qingdao, China. Her current research interests are process mining, Petri nets and workflow.



Na Su received her B.Sc. degree from the Liaocheng Normal University, Shangdong, China, in 2000, her M.Sc. degree from the Qingdao University of Science and Technology, Qingdao, China, in 2015. She is currently Associate Professor at the College of Qingdao Huanghai University, Qingdao, China. Her current research interests are process mining, Petri nets and workflow.



Yu Liang received his Bachelor degree in computer science and technology from the Shandong Agricultural University, China in 2015. He received his Master degree in software engineering from the Shandong University of Science and Technology, China in 2019. He is currently pursuing Doctorate in the Department of Computer Science and Technology, Tongji University, Shanghai. His research interests include lightweight deep neural networks (DNNs), interpretation methods for DNNs and graph DNNs.

Liang Qi received his B.Sc. degree in information and computing science and his M.Sc. degree in computer software and theory from the Shandong University of Science and Technology, Qingdao, China, in 2009 and 2012, respectively, and his Ph.D. degree in computer software and theory from the Tongji University, Shanghai, China in 2017. From 2015 to 2017, he was Visiting Student with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA. He is currently Associate Professor with the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China. He has over 80 papers in journals and conference proceedings, including the IEEE Transactions on Intelligent Transportation Systems, the IEEE/CAA Journal of Automatica Sinica, the IEEE Transactions on System, Man and Cybernetics: Systems, the IEEE Transactions on Computational Social Systems, the IEEE Transactions on Automation Science and Engineering, the IEEE Transactions on Cybernetics, the IEEE Transactions on Network Science and Engineering, IEEE Transactions on Image Processing, and IEEE Signal Processing Letters. He received the Best Student Paper Award-Finalist in the 15$^{\text{th}}$ IEEE International Conference on Networking, Sensing and Control (ICNSC 2018). His current research interests include Petri nets, optimization algorithms, machine learning, and intelligent transportation systems.

Yuyue Du received the B.Sc. degree from the Shandong University, Jinan, China, in 1982, his M.Sc. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 1991, and his Ph.D. degree in computer application from the Tongji University, Shanghai, China, in 2003. He is currently Professor at the College of Information Science and Engineering, Shandong University of Science and Technology, Qingdao, China. He has taken in over 10 projects supported by the National Nature Science Foundation, the National Key Basic Research Developing Program, and other important and key projects at provincial levels. He has published over 200 papers in domestic and international academic publications. His research interests are in formal engineering, Petri nets, real-time systems, process mining, and workflows.