Computing and Informatics, Vol. 45, 2026, 154–181, doi: 10.31577/cai_2026_1_154

LPN-BASED MODEL REPAIR METHOD FOR CHANGED BUSINESS PROCESSES

Yuyue Du

College of Computer, Shandong Xiehe University
6277 Jiqing Road, Licheng District, 250109 Jinan City, China
e-mail: yuyuedu@sdust.edu.cn

Yuhua Xu*

School of Information and Intelligent Science, Donghua University
2999 North Renmin Road, Songjiang District, 201620 Shanghai City, China
e-mail: xuyuhua@dhu.edu.cn

Liang Qi, Yijia Liu

College of Computer Science and Engineering
Shandong University of Science and Technology
579 Qianwangang Road, Huangdao District, 266590 Qingdao City, China
e-mail: qiliang@sdust.edu.cn, 202311070212@sdust.edu.cn

Qun Wang, Mei Chen

College of Computer, Shandong Xiehe University
6277 Jiqing Road, Licheng District, 250109 Jinan City, China
e-mail: wangqun@sdxiehe.edu.cn, wangxiaoyan2@sdxiehe.edu.cn

Abstract. Information systems play an important role in realizing business processes. Process mining uses event logs produced by information systems to construct

* Corresponding author

the models of business processes. Then, the business process model can be used to verify and improve business processes. To represent a changing business process, some structures of the process model need to be replaced with other structures. However, existing model repair methods do not consider replacing the structures of the process model. This work proposes a logical Petri net-based approach to repair models based on the order relations among activities. First, we construct two relation sets of an event log and a process model, respectively. Comparing the two relation sets, the differences between a process model and an event log are collected in a deviation set. According to the deviation set, the model is repaired by constructing concurrent structures. Finally, we perform some experiments to illustrate the effectiveness and correctness of the proposed approach. The results show that the proposed method produces a more precise and simpler repaired model compared to state-of-the-art methods.

Keywords: Business process, process mining, logical Petri nets, model repair, concurrent structures

1 INTRODUCTION

Business process become more and more complex due to the development of information systems. To understand business processes, we can model and analyze them by extracting information from systems [1, 2]. Process mining is based on process model-driven methods and data mining. It establishes two connections, one is between a process model and a business process, and one is between the business process and its data. It can discover, verify, detect, and improve business processes' models based on the information contained in event logs [3]. Process discovery [3], conformance checking [4], and process enhancement [5] are its three objectives.

Process discovery techniques use event logs to automatically construct process models. Researchers have proposed some process discovery algorithms in recent years. For instance, van der Aalst et al. design an α algorithm [6], which can build models according to the order relations among activities. However, it cannot effectively obtain a model containing repetitive activities, invisible transitions or some complex structures. Therefore, many variations of α algorithm have been designed. For example, $\alpha\#$ algorithm proposed by Wen et al. [7] can mine invisible transitions. $\alpha{+}{+}$ algorithm in [8] can mine non-free choice structures effectively. And Weber et al. propose a framework for analyzing process mining algorithms [9].

The qualities of a mined process model can be measured in four metrics: fitness, generalization, simplicity, and precision [3, 10]. Fitness requires a model to replay traces (a finite sequence of activities) in event logs. Generalization allows the process model to contain some activities that happen in the future. Simplicity denotes that the model that can replay all traces in event logs should have a small

net size. Precision requires that the occurrence of activities that are not contained in the event logs is not allowed in a process model.

Conformance checking compares a model with an event log to check whether a model is consistent with a real-life business process. Replaying the event log on the model, it finds the differences between them. Therefore, we can detect where the process model needs to be changed by conformance check techniques. Existing conformance checking approaches are mainly alignment, footprint comparison, token replaying [3], and mirroring matrices.

Process enhancement uses event logs generated by an actual business process to improve or expand an existing model. When an original process model cannot represent a changed business process correctly, one solution is to build a new model using some process discovery algorithms [6, 7, 8, 9]. However, the new model often has low similarity to the original one and loses the original one's advantages. A better solution is to repair the original model such that it can accurately represent the business process and replay event logs. In this paper, we introduce model repair, which is a new process enhancement technology. When a process model does not correctly replay event logs, it needs to be repaired according to the results of conformance checking. The repaired model can preserve the sub-model that can replay event logs. Therefore, the similarity between the two models can be ensured.

Currently, several methods for repair have been proposed. For example, Fahland and van der Aalst [11] (called the FA-method in this paper) detect all differences (called deviations in this paper) between a process model and an event log based on optimal alignments. A non-fitting sub-log can be collected according to the results of alignments. To replay the non-fitting sub-log, it constructs a corresponding sub-process, and adds it to a suitable position of the original model. Besides, a loop that can replay a non-fitting sub-log can also be discovered. The method is the first study of the model repair technique. It ensures that the repaired model's fitness value is high. Polyvyanyy et al. [12] mainly consider limiting the consumption of resources when repairing process models, and put forward several methods. For instance, the G-method repairs the model by performing two types of operations: skipping an activity or adding a new activity as a self-loop in the model. This method can also guarantee a high fitness value of the repaired model. However, the methods in [11, 12] add sub-processes or self-loops to the model, which can generate repetitive transitions and make the model complex and redundant. Other recent model repair methods [13, 14, 15, 16] focus on some special structures or cases. Sun et al. [13] design an approach to repair workflows based on mirroring matrices. It firstly identifies model deviation domains by comparing mirroring matrices. Then, the model is changed by inserting new activities into the actual process. Qi et al. [14] put forward an approach to repair choice structures of models. First, they detect all deviations between an original model and an event log based on extended alignments. Then it inserts new branches into the model's choice structures. In [15], some methods are proposed to add bridges among different choice branches. Not only can activities on the same choice branch

be replayed by the repaired model, but also activities on different branches can be replayed. A method for repairing process models with selection structures is proposed in [16]. According to the relationship between the input and output places of a sub-model, the deviation position is determined by a token replay method. Then, some algorithms are designed to repair the process models based on Petri nets.

Some changes to a business process easily tend to allow activities to be executed concurrently, which were not studied in the past. Therefore, the related structures in such process models may need to be replaced, such as changed choice structures or sequential structures to concurrent structures. The model repair methods in [13, 14, 15, 16] have good results for some special structures or cases, but they cannot replace the structures in the models. Other existing Petri net-based model repair approaches [11, 12] usually add invisible transitions or self-loops to models. They do not guarantee a high simplicity and precision of the repaired model. Besides, the order relations among activities in event logs may not be represented correctly by the repaired model. To solve the above problems, this work for the first time proposes a model repair method to replace choice or sequential structures with concurrent structures. Logical Petri nets (LPNs) [17, 18, 19] are an advanced type of Petri nets that can help make models simpler and more precise and accurately represent real business processes. Here, the repaired model is described by LPN. The LPN-based repaired model can correctly depict activities' relations, i.e., an actual business process can be correctly described. This work makes the following contributions:

1. We construct a log order set to record activities' relations in an event log, and a model order set to collect transitions' relations in a process model. Then, a new conformance checking method is introduced to detect all deviations between a process model and an event log.
2. According to the deviation set, two LPN-based methods are designed to repair the original model. Choice and sequential structures in the model can be changed to concurrent structures by these two methods.
3. Simulated experiments are conducted on the data of the treatment process of tumor patients in a hospital. The results of experiments show the proposed methods' effectiveness and correctness.

The rest of this paper is organized as follows. Some basic notions about the process models, Petri nets, and logical Petri nets are introduced in Section 2. Section 3 proposes a new conformance checking method to detect deviations between an original model and an event log. Section 4 proposes two LPN-based model repair methods that can change choice structures and sequential structures to concurrent structures, respectively. Section 5 conducts some experiments. Section 6 gives a conclusion about this work.

2 PRELIMINARIES

Petri nets [20, 21, 22, 23, 24, 25] can describe and analyze the information systems with concurrent, asynchronous, distributed, and uncertain properties. LPNs [17] as a kind of advanced Petri nets can improve models' simplicities and precisions. This section briefly introduces their definitions [18, 19, 26, 27, 28, 29, 30], event logs, process mining [3, 5, 31], and process models [32].

Definition 1 (Sequences)**.** Let A be a set. A sequence u over A is denoted as $u = \langle u[1], u[2], \ldots, u[n] \rangle$, where $|u| = n$ is the length of u, and $u[i] \in A$ $(1 \leq i \leq n)$ refers to the i^{th} element of u.

Definition 2 (Multiple sets)**.** Let A be a set. A function D: $A \to \mathbb{N}$ is called a multiple set over A, where $\mathbb{N} = \{0, 1, 2, \ldots\}$. Notation $\mathcal{B}(A)$ denotes the set of all multiple sets of A.

Definition 3 (Traces and event logs)**.** Let A be a set of all activities. A^* denotes a finite sequence set on A. A trace $\sigma \in A^*$ is a sequence of activities. $\varsigma(\sigma)$ represents all activities in σ. An event log $L \in \mathcal{B}(A^*)$ is a multiple set of traces.

Definition 4 (Pre- and post-activity sets)**.** Given $L \in \mathcal{B}(A^*)$, $\forall \nu \in A$, we have

$$^{\blacklozenge}\nu = \{\omega \mid \omega \in A, \sigma \in L, \sigma[i-1] = \omega, \sigma[i] = \nu, 1 \leq i \leq |\sigma|\}, \tag{1}$$

$$\nu^{\blacklozenge} = \{\omega \mid \omega \in A, \sigma \in L, \sigma[i] = \nu, \sigma[i+1] = \omega, 1 \leq i \langle |\sigma|\}, \tag{2}$$

where $^{\blacklozenge}\nu$ and $\nu^{\blacklozenge}$ are the pre- and post-activity sets of ν in L, respectively.

For instance, let $A' = \{a, b, c, d\}$ and $L' = \{\sigma_1, \sigma_2\} = \{\langle a, b, c, e\rangle, \langle a, b, c, e\rangle\}$. For $b \in A'$, we have $^{\blacklozenge}c = \{b\}$ and $c^{\blacklozenge} = \{d, e\}$.

Definition 5 (Input and output sets)**.** $N = (P, T, F)$ is a net, where P, T, and $F \subseteq (P \times T) \cup (T \times P)$ are the finite sets of places, transitions, and arcs, respectively. $\forall y \in P \cup T$, we have

$$^{\bullet}y = \{z \mid z \in T \cup P, (z, y) \in F\}, \tag{3}$$

$$y^{\bullet} = \{z \mid z \in T \cup P, (y, z) \in F\}, \tag{4}$$

where $^{\bullet}y$ and $y^{\bullet}$ are the input and output sets of y in N, respectively.

Definition 6 (Petri net)**.** A Petri net is a four tuple $PN = (P, T; F, M)$, where

1. $N = (P, T; F)$ is a net;
2. $M : P \to \mathbb{N}$ is a marking function that defines the number of tokens in p $(p \in P)$; and
3. Transition firing rules are that:

(a) For $t \in T$, if $\forall p \in {}^\bullet t : M(p) \geq 1$, $M[t\rangle$, i.e., t is enabled at M; and
(b) If $M[t\rangle$, then $M[t\rangle M'$, i.e., t can be fired and a new marking M' yields from M, where

$$M'(p) = \begin{cases} M(p) - 1, & p \in {}^\bullet t - t^\bullet, \\ M(p) + 1, & p \in t^\bullet - {}^\bullet t, \\ M(p), & \text{else.} \end{cases} \quad (5)$$

A block-structured Petri net includes four structures: choice, sequential, loop, and concurrent structures [32]. It contains an original place $p_o \in P$ and a final place $p_f \in P$, i.e., ${}^\bullet p_o = \emptyset$ and $p_f^\bullet = \emptyset$. $\forall y \in T \cup P$ is on some path from p_o to p_f.

Definition 7 (Process model)**.** $PM = (PN, \lambda, M_o, M_f)$ is a process model, where

1. $PN = (P, T; F, M)$ is a block-structured Petri net;
2. λ: $T \to A \cup \{\tau\}$ is a function that matches a transition to an activity, and τ denotes an invisible transition; and
3. M_o is the original marking, and M_f is the final marking.

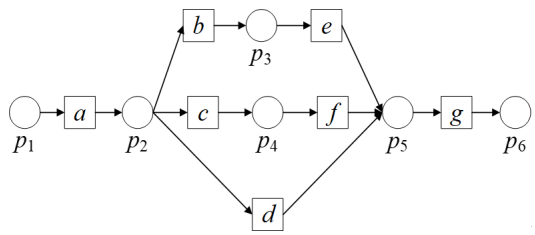

Figure 1. A Process model PM_1

In this paper, we only consider process models. For illustration, we use activities to describe transitions. A process model PM_1 is shown in Figure 1 where p_1 is an initial place and p_6 is a final one.

Definition 8 (Complete firing sequence)**.** Given a process model $PM = (PN, \lambda, M_o, M_f)$, a transition sequence $s \in T^*$ satisfying $M_o[s\rangle M_f$ is called a complete firing sequence. The set S_{PM} records all complete firing sequences in PM.

Definition 9 (Logical Petri net)**.** $LPN = (P, T; F, I, O, M)$ is called a logical Petri net, where

1. P is a finite set of places;
2. $T = T_I \cup T_O \cup T_M$ is a finite set of transitions, and $\forall t \in T_I \cap T_O$: $t^\bullet \cap {}^\bullet t = \emptyset$, where

   (a) T_I represents a set of logical input transitions, $\forall t \in T_I$, the logical input function $f_I(t)$ restricts ${}^\bullet t$;

(b) T_O denotes a set of logical output transitions, $\forall t \in T_O$, the logical output function $f_O(t)$ restricts ${}^\bullet t$;
(c) T_D denotes a set of ordinary transitions as defined in Definition 6;

3. $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs;
4. I maps a logical input transition to a logical input function, and $\forall t \in T_I$, we have $I(t) = f_I(t)$;
5. O maps a logical output transition to a logical output function, and $\forall t \in T_O$, $O(t) = f_O(t)$;
6. M: $P \to \{0, 1\}$ denotes a marking function, and $\forall p \in P$, $M(p)$ denotes the number of tokens in p;
7. Transition firing rules are that:

(a) $\forall t \in T_I$ is enabled only if $f_I(t)|_M = {}_\bullet T_\bullet$, $M[t\rangle M'$, where $M'(p) = 0$ if $p \in {}^\bullet t$, $M'(p) = 1$ if $p \in t^\bullet$ and $M'(p) = M(p)$ if $p \notin t^\bullet \cup {}^\bullet t$;
(b) $\forall t \in T_O$ fires only $\forall p \in {}^\bullet t$ satisfies $M(p) = 1$, $M[t\rangle M'$, where $M'(p) = 0$ if $p \int^\bullet$, $f_O(t)|_M = {}_\bullet T_\bullet$ is satisfied by $\forall p \in t^\bullet$, and $M'(p) = M(p)$ if $p \notin t^\bullet \cup {}^\bullet t$; and
(c) $\forall t \in T_D$, it has the same firing rules as those in Petri nets.

Similarly, S_{LPN} is a complete firing sequence set of *LPN*.

Logical Petri nets is an extention of classical Petri nets where the firing of an input transition is restricted by the tokens of its input places which satisfy a logical expression of the input places, while the firing of an output transition restrict the generated tokens of the output places which satisfy a logical expression. A logical Petri net denoted by LPN' is shown in Figure 2, where a is a logical input transition, c is a logical output one, and others are traditional ones. $I(a) = p_2 \wedge (p_1 \otimes p_3)$ represents the logical input function of a. There are two cases before a fires:

1. both p_1 and p_2 contain a token;
2. both p_2 and p_3 contain a token.

The logical output function of c is $O(c) = p_6 \vee p_7$. There are three cases after c is fired:

1. only p_6 contains a token;
2. only p_7 contains a token; or
3. both p_6 and p_7 contain a token.

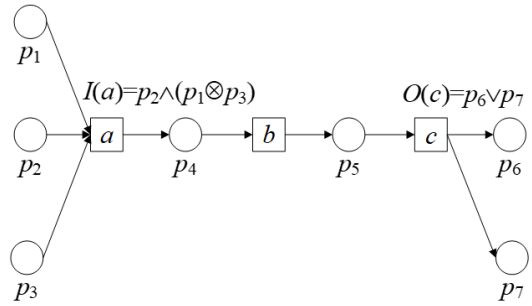

Figure 2. A logical Petri net LPN_1

3 DEVIATION SET

As an actual situation changes, a business process may be changed to reflect the changed situation. For new event logs generated by a real business process, the original model needs to be checked if it can replay these new event logs. Then we can confirm whether a process model is consistent with its current business process or not. In this section, a new conformance checking method of collecting all deviations between a process model and event logs is proposed.

3.1 Order Relation Sets of Event Log and Model

To accurately describe activities' order relations in an event log, we give the following concepts ground on [33, 34].

Definition 10 (Added order relations)**.** Given an event log $L \in \mathcal{B}(A^*)$, $\forall \alpha, \beta \in A$, we have

1. Adjacent relation: $\alpha > \beta$ only if $\exists \sigma \in L$, $1 \leq i < |\sigma|$: $\sigma[i] = \alpha$ and $\sigma[i+1] = \beta$;
2. Causal relation: $\alpha \rightarrow \beta$ only if $\exists \alpha > \beta$ and $\nexists \beta > \alpha$;
3. Choice relation: $\alpha \times \beta$ only if $\forall \sigma \in L$: $\alpha \in \varsigma(\sigma)$ and $\beta \notin \varsigma(\sigma)$, and vice versa;
4. Classic concurrent relation: $\alpha || \beta$ only if $\exists \sigma_1, \sigma_2 \in L$: $\alpha > \beta$ in σ_1 and $\beta > \alpha$ in σ_2; and
5. Logical concurrent relation: $a \vee b$ only if $\exists \sigma_1, \sigma_2 \in L$: $\alpha || \beta$, $\exists \sigma_3 \in L$: $\alpha \in \varsigma(\sigma_3)$ and $\beta \notin \varsigma(\sigma_3)$, and vice versa.

Definition 11 (Log order set)**.** Given $L \in \mathcal{B}(A^*)$, $R_L = \{\alpha \otimes \beta | \alpha, \beta \in A\}$ is called a log order set, where $\otimes \in \{\rightarrow, \times, ||, \vee\}$ represents the order relation based on L between α and β.

According to Definition 11, we can get a log order set of L denoted by R_L. $\forall L \in \mathcal{B}(A^*)$ and α, $\beta \in A$, there are four possible relations between α and β:

$\alpha \to \beta$, $\alpha \times \beta$, $\alpha||\beta$, or $\alpha \vee \beta$. Among them, $\alpha \to \beta$ is a causal relation, i.e., α is followed directly by β, but β is never followed directly by α in L. $\alpha \times \beta$ is a choice relation, i.e., α and β are not included in the same trace. $\alpha||\beta$ is a classic concurrent relation, i.e., α is followed directly by β in one trace, and β is followed by α in another trace. $\alpha \vee \beta$ is the logical concurrent relation, i.e., there exists three traces $\sigma_1, \sigma_2, \sigma_3 \in L$, we have $\alpha||\beta$ according to traces σ_1 and σ_2, and only one of α and β is contained in σ_3.

According to Definitions 10 and 11, Algorithm 1 is used to collect the activities' order relations in an event log.

Algorithm 1: Compute Log Order Set

Input: An event log $L \in \mathcal{B}(A^*)$
Output: The log order set of L represented by R_L

1 $R = \emptyset$, $R_L = \emptyset$;
2 **for** $\forall \sigma \in L$: $\alpha_i \in \&(\sigma)$ where $1 \leq i < |\sigma|$ do
3 if $\alpha_i > \alpha_{i+1} \notin R$, then
4 R=$R \cup \{\alpha_i > \alpha_{i+1}\}$;
5 **end for**
6 **for** $\alpha > \beta \in R$, if there is no $\beta > \alpha \in R$, then
7 $R_L = R_L \cup \{\alpha \to \beta\}$;
8 **end for**
9 **for** $\forall \sigma \in R$: if $\alpha \in \varsigma(\sigma)$ and $\beta \notin \varsigma(\sigma)$, and vice versa, then
10 $R_L = R_L \cup \{\alpha \times \beta\}$;
11 **end for**
12 **for** $\alpha > \beta \in R$ and $\beta > \alpha \in R$ do
13 if $\forall \sigma \in L$: $\alpha \in \varsigma(\sigma)$ and $\beta \in \varsigma(\sigma)$, then
14 $R_L = R_L \cup \{\alpha||\beta\}$;
15 else if $\exists \sigma \in L$: $\alpha \in \varsigma(\sigma)$ and $\beta \notin \varsigma(\sigma)$, and vice versa, then
16 $R_L = R_L \cup \{\alpha \vee \beta\}$;
17 **end for**
18 **return** R_L

Algorithm 1 first initializes R and R_L at Step 1. According to Steps 2–5, all adjacent relations among activities in L can be found. Steps 6–8 collect all causal relations. Steps 9–11 are used to collect all choice relations. According to Steps 12–14 and 15–17, we can collect all classic and logical concurrent relations in L, respectively. Finally, the log order set R_L is obtained to record all activities' order relations in L.

Here, the execution process of Algorithm 1 is illustrated by the following example. Let $L_1 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7, \sigma_8, \sigma_9, \sigma_{10}, \sigma_{11}, \sigma_{12}\} = \{\langle a, b, e, g\rangle, \langle a, c, f, g\rangle, \langle a, d, g\rangle, \langle a, b, e, c, f, g\rangle, \langle a, b, c, e, f, g\rangle, \langle a, b, c, f, e, g\rangle, \langle a, c, b, e, f, g\rangle, \langle a, c, b, f, e, g\rangle, \langle a, c, f, b, e, g\rangle, \langle a, b, e, d, g\rangle, \langle a, b, d, e, g\rangle, \langle a, d, b, e, g\rangle\}$, then the log order set of L_1 is $R_{L1} = \{a \to b, a \to c, a \to d, b \to e, e \to g, c \to f, f \to g, d \to g, c \times d, d \times f, b \vee$

$c, b \vee d, b \vee f, c \vee e, d \vee e, e \vee f\}$.

The added order relations can also be used to find relations among transitions in the corresponding model. According to Definitions 10 and 11 and Algorithm 1, the order relation set of a process model is defined as follows.

Definition 12 (Model order set)**.** Given a process model $PM = (PN, \lambda, M_o, M_f)$, and a complete firing sequence set S_{PM} of PM. $R_M = \{t_i \otimes' t_j | t_i, t_j \in T\}$ is called a model order set, where $\otimes' \in \{\rightarrow, \times, ||\}$ denotes the order relation based on S_{PM} between transitions t_i and t_j.

We now define the order relation set of LPN.

Definition 13 (Logical model order set)**.** Given a logic Petri net $LPN = (P, T; F, I, O, M)$, and a complete firing sequence set S_{LPN} of LPN, $R_{LM} = \{t_i \otimes t_j | t_i, t_j \in T\}$ is called a logical model order set, where $\otimes \in \{\rightarrow, \times, ||, \vee\}$ denotes the order relation based on S_{LPN} between transitions t_i and t_j.

Note that, Petri nets cannot describe logic concurrent relations among transitions correctly. The process model PM_1 depicted in Figure 1 has three complete firing sequences: $s_1 = \langle a, b, e, g\rangle$, $s_2 = \langle a, c, f, g\rangle$, and $s_3 = \langle a, d, g\rangle$. Then we can obtain the model order set of PM_1 according to Algorithm 1: $R_{M1} = \{a \rightarrow b, a \rightarrow c, a \rightarrow d, b \rightarrow e, e \rightarrow g, c \rightarrow f, f \rightarrow g, d \rightarrow g, b \times c, b \times d, b \times f, c \times d, c \times e, d \times e, d \times f, e \times f\}$.

3.2 Deviations Between Model and Event Log

R_L records all activities' order relations in an event log, and R_M records all transitions' order relations in a process model. Comparing R_L with R_M, all deviations between a process model and an event log can be found. Thus, we have the following definition to compute these deviations.

Definition 14 (Deviation set)**.** Given a model order set R_M and a log order set R_L, R_D is called a deviation set containing three types of elements

1. if $\alpha \otimes \beta \in R_L$, $\alpha \otimes' \beta \in R_M$, and $\otimes \neq \otimes'$, $\alpha \otimes \beta | \alpha \otimes' \beta \in R_D$;
2. if $\alpha \otimes \beta \in R_L$ and $\alpha \otimes' \beta \notin R_M$, and $\alpha \otimes \beta | \emptyset \in R_D$; and
3. if $\alpha \otimes \beta \notin R_L$ and $\alpha \otimes' \beta \in R_M$, $\emptyset | \alpha \otimes' \beta \in R_D$.

If $\alpha \otimes \beta \in R_L$, $\alpha \otimes' \beta \in R_M$, and $\otimes = \otimes'$, i.e., the order relation between α and β in L is same as in PM, we do not add any element to R_D. In this paper, R_D(a) represents the number of elements containing a in R_D, and A_{RD} denotes the set of activities contained in R_D.

We can see that the differences between R_L and R_M are collected by R_D. From the above definition, there are three reasons when a model PM does not conform to an event log L:

1. some activities' order relations in L are different from those in PM;
2. some activities' order relations in L cannot be depicted in PM; and
3. some transitions' order relations in PM do not appear in L.

We compute the deviation set according to Algorithm 2. It can find all deviations between an original process model and an event log.

Algorithm 2: Deviation Computation Algorithm

Input: R_L and R_M
Output: The set of deviations denoted by R_D

1 $R_D = \emptyset$;
2 **for** $\forall \alpha \otimes \beta \in R_L$ do
3 **if** $\exists \alpha, \beta$, such that $\alpha \otimes' \beta \in R_M$ and $\otimes = \otimes'$, then
4 $R_D = R_D$
5 **else if** $\exists \alpha, \beta$ such that $\alpha \otimes' \beta \in R_M$ and $\otimes \neq \otimes'$, then
6 $R_D = R_D \cup \{\alpha \otimes \beta \mid \alpha \otimes' \beta\}$;
7 **else**
8 $R_D = R_D \cup \{\alpha \otimes \beta \mid \emptyset\}$;
9 **end for**
10 **for** $\forall \alpha \otimes' \beta \in R_M$ do
11 if $\nexists \alpha, \beta$ such that $\alpha \otimes \beta \in R_L$, then
12 $R_D = R_D \cup \{\emptyset \mid \alpha \otimes' \beta\}$;
13 **end for**
14 **return** R_D

Algorithm 2 is described as follows. R_D is initialized by Step 1. Steps 2–13 add elements to R_D. Finally, the set of deviations between L and PM can be obtained.

We now give an example to illustrate Algorithm 2. According to the above examples, $R_{D1} = \{b \vee c | b \times c, b \vee d | b \times d, b \vee f | b \times f, c \vee e | c \times e, d \vee e | d \times e, e \vee f | e \times f\}$, where $R_{D1}(b) = 3, R_{D1}(c) = 2, R_{D1}(d) = 2, R_{D1}(e) = 3, R_{D1}(f) = 2$, and $A_{RD1} = \{b, c, d, e, f\}$. According to R_{D1}, all deviations between L_1 and PM_1 can be found. Next, we repair a model according to a deviation set.

4 CONSTRUCTING AND USING CONCURENT STRUCTURES TO REPAIR MODELS

With the business process changing or being updated, a process model may not replay new event logs correctly. When the corresponding model cannot depict some activities' order relations in event logs, some structures in the model need to be replaced with other structures such that a repaired model can accurately describe activities' relations. Before repairing the model, we need to identify deviation activities in R_D.

Definition 15 (Deviation activity)**.** Let R_D be a deviation set between L and PM. $\alpha \in A_{RD}$ is called a deviation activity if $\forall \beta \in A_{RD}$: $R_D(\alpha) \geq R_D(\beta)$.

The structures which have transitions corresponding to deviation activities need to be changed. As an actual business process is updated, some activities can occur concurrently now, but not in the past. We then give a repair method if the transitions corresponding to deviation activities are in sequential or choice structures, while these activities in event logs have logical concurrent relations. Thus, these structures in the model should be repaired to concurrent ones. We propose two algorithms to respectively change choice and sequential structures to concurrent ones.

4.1 Model Repair for Choice Structures

Some activities in an event log show logical concurrent relations after a real process changes or is updated. These activities can be concurrently executed. However, only one of these activities can fire in an actual model, i.e., these activities are part of a choice structure. At this time, we should repair the model by changing the choice structure to a concurrent one.

Now, we use Algorithm 3 to repair the model, and it can change choice structures to concurrent ones.

Algorithm 3 changes choice structures to concurrent ones. LPN' is initialized by Step 1. According to Steps 2–3, R_L, R_M, R_D, and R'_D can be obtained. According to Steps 4–6, $\forall \alpha \vee \beta \mid \alpha \times \beta \in R'_D$, if the input sets (output sets) of α and β do not have the same element, we compute $R'_D = R'_D - \{\alpha \vee \beta \mid \alpha \times \beta\}$, i.e., removing $\alpha \vee \beta \mid \alpha \times \beta$ from R'_D. $\forall \alpha \vee \beta \mid \alpha \times \beta \in R'_D$, it has the following three cases:

1. If α and β have the same input sets, i.e., ${}^\bullet\alpha = {}^\bullet\beta$ and $\alpha^\bullet \neq \beta^\bullet$, then we execute Steps 8–17. $R_D(\alpha) \geq R_D(\beta)$ denotes that α is a deviation activity. At this time, ${}^\bullet({}^\bullet\beta)$ is considered to be a logical output transition. According to Steps 10–11, we delete some original arcs, and insert some new arcs and places into the model. The logical output function of ${}^\bullet({}^\bullet\beta)$ is computed by Step 12. Steps 13–16 are similar to Steps 9–12;
2. If α and β have the same output sets, i.e., $\alpha^\bullet = \beta^\bullet$ and ${}^\bullet\alpha \neq {}^\bullet\beta$, then we execute Steps 18–27; and
3. If ${}^\bullet\alpha = {}^\bullet\beta$ and $\alpha^\bullet = \beta^\bullet$, Steps 28–30 are executed. Finally, PM is repaired to LPN'.

Theorem 1. Let L be an event log, $PM = (PN, \lambda, M_o, M_f)$ be a process model, and $LPN' = (P', T'; F', I', O', M')$ be a logical Petri net repaired by Algorithm 3. $\forall \alpha \vee \beta \mid \alpha \times \beta \in R'_D$, if $R_D(\alpha) \geq R_D(\beta)$, then ${}^\bullet\alpha = {}^\bullet\beta$ ($\alpha^\bullet = \beta^\bullet$) in PM and ${}^\bullet\alpha \neq {}^\bullet\beta$ ($\alpha^\bullet \neq \beta^\bullet$) in LPN'.

Algorithm 3: Choice Structures-based Model Repair Algorithm

Input: A process model PM and an event log L
Output: The repaired model $LPN' = (P', T'; F', I', O', M')$

1 $LPN' = PM$
2 Executing Algorithm 1 to compute R_L and R_M;
3 Executing Algorithm 2 to compute R_D, and $R'_D = R_D$;
4 **for** $\forall \alpha \vee \beta | \alpha \times \beta \in R'_D$: if ${}^\bullet\alpha \neq {}^\bullet\beta$ or $\alpha^\bullet \neq \beta^\bullet$, then
5 $R'_D = R'_D - \{\alpha \vee \beta | \alpha \times \beta\}$; //$\alpha \vee \beta | \alpha \times \beta$ means $\alpha \vee \beta$ in R_L, $\alpha \times \beta$ in R_M;
6 **end for**
7 **for** $\forall \alpha \vee \beta | \alpha \times \beta \in R'_D$ do // $\alpha \vee \beta | \alpha \times \beta$ is the same as $\beta \vee \alpha | \beta \times \alpha$;
8 **if** (${}^\bullet\alpha = {}^\bullet\beta$ and $\alpha^\bullet \neq \beta^\bullet$), then
9 if ($R_D(\alpha) \geq R_D(\beta)$), then
10 $P' = P' \cup \{p_s\}$ and $p_s = {}^\bullet\alpha$;
11 $F' = F' - \{{}^\bullet\beta \to \alpha\} \cup \{{}^\bullet({}^\bullet\beta) \to p_s, p_s \to \alpha\}$;
12 $O' = O' \cup \{O'({}^\bullet({}^\bullet\beta)) = {}^\bullet\beta \vee p_s\}$;
13 else
14 $P' = P' \cup \{p_s\}$ and $p_s = {}^\bullet\beta$;
15 $F' = F' - \{{}^\bullet\alpha \to \beta\} \cup \{{}^\bullet({}^\bullet\alpha) \to p_s, p_s \to \beta\}$;
16 $O' = O' \cup \{O'({}^\bullet({}^\bullet\alpha)) = {}^\bullet\alpha \vee p_s\}$;
17 **end if**
18 **else if** ($\alpha^\bullet = \beta^\bullet$ and ${}^\bullet\alpha \neq {}^\bullet\beta$), then
19 if ($R_D(\alpha) \geq R_D(\beta)$), then
20 $P' = P' \cup \{p_t\}$ and $p_t = \alpha^\bullet$;
21 $F' = F' - \{\alpha \to \beta^\bullet\} \cup \{\alpha \to p_t, p_t \to (\beta^\bullet)^\bullet\}$;
22 $I' = I' \cup \{I'((\beta^\bullet)^\bullet) = \beta^\bullet \vee p_t\}$;
23 else
24 $P' = P' \cup \{p_t\}$ and $p_t = \beta^\bullet$;
25 $F' = F' - \{\beta \to \alpha^\bullet\} \cup \{\beta \to p_t, p_t \to (\alpha^\bullet)^\bullet\}$;
26 $I' = I' \cup \{I'((\alpha^\bullet)^\bullet) = \alpha^\bullet \vee p_t\}$;
27 **end if**
28 **else if** (${}^\bullet\alpha = {}^\bullet\beta$ and $\alpha^\bullet = \beta^\bullet$), then
29 Go to Steps 9–16 and 19–26;
30 **end if**
31 **end for**
32 **return** LPN'

Proof. For $\alpha \vee \beta \mid \alpha \times \beta \in R'_D$, we have $\alpha \vee \beta \in R_L$ and $\alpha \times \beta \in R_M$ according to Definition 14. $R_D(\alpha) \geq R_D(\beta)$, i.e., α is a deviation activity.

1. $\exists x \in A$ and $x \in T$: for $\alpha \times \beta \in R_M$, $x = {}^\bullet({}^\bullet\alpha) = {}^\bullet({}^\bullet\beta)$, and $\exists p \in P$: $p = {}^\bullet\alpha = {}^\bullet\beta$ in PM; for $\alpha \vee \beta \in R_L$, $x = {}^\blacklozenge\alpha = {}^\blacklozenge\beta$ in L. Because $\alpha \vee \beta | \alpha \times \beta \in R'_D$, a new place p_s ($p_s = {}^\bullet\alpha$) is added between α and x. Thus, ${}^\bullet\beta \neq {}^\bullet\alpha$ in LPN'.
2. $\exists e \in A$ and $e \in T$: for $\alpha \times \beta \in R_M$, $e = (\alpha^\bullet)^\bullet = (\beta^\bullet)^\bullet$, and $\exists p' \in P$ satisfies $p' = \alpha^\bullet = \beta^\bullet$ in PM; for $\alpha \vee \beta \in R_L$, $e = \alpha^\blacklozenge = \beta^\blacklozenge$ in L. Because $\alpha \vee \beta | \alpha \times \beta \in R'_D$, a new place p_t ($p_t = \alpha^\bullet$) is added between α and e. Thus, $\beta^\bullet \neq \alpha^\bullet$ in LPN'.

□

We illustrate Algorithm 3 by an example. The event log L_1 is mentioned above. The process model denoted by PM_1 is depicted in Figure 1. Now, we repair PM_1 according to L_1, and the repaired model based on the proposed method is depicted in Figure 3. This model only adds two arcs and two places. The repaired model based on the method [11] is shown in Figure 4. To replay event log L_1, this method inserts ten arcs and five invisible transitions into PM_1. Figure 5 shows the model repaired based on the method [12]. Comparing Figures 1 and 5, it can be seen that the repaired model is added twelve arcs and six repetitive transitions. Both models (Figures 4 and 5) are much more complex than ours (Figure 3).

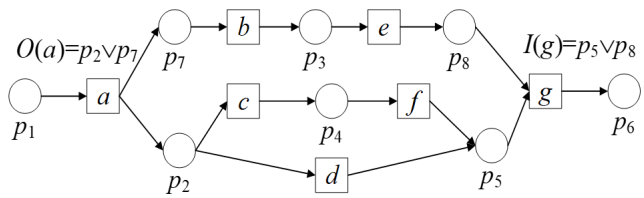

Figure 3. The repaired model based on our method

4.2 Model Repair for Sequential Structures

After a business process changes, some activities can be executed sequentially and concurrently. Therefore, we need to replace the sequential structure with a concurrent one in the original model. To solve this problem, we design Algorithm 4.

Algorithm 4 changes sequential structures of models to concurrent ones. LPN'' is initialized by Step 1. We can get R_L, R_M, and R_D according to Steps 2–3. Steps 4–6 delete some original arcs from F''. According to the different elements in R_D, Steps 7–19 insert new arcs and places into the model, and compute logical functions. Finally, PM is repaired to LPN''.

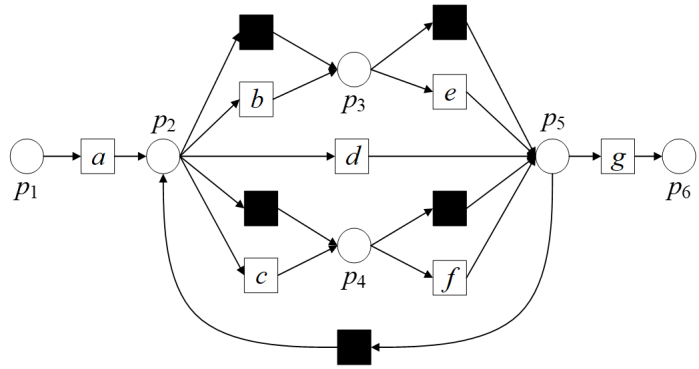

Figure 4. The repaired model based on the method [11]

Theorem 2. Let L be an event log, $PM = (PN, \lambda, M_o, M_f)$ be a process model, and $LPN'' = (P'', T''; F'', I'', O'', M'')$ be a logical Petri net repaired by Algorithm 4. For $\forall a \to b \mid \emptyset \in R_D$,

1. If $\exists \beta \vee x \mid \emptyset \in R_D$ or $\beta \vee x \mid x \to \beta \in R_D$, ${}^\bullet\beta \in P'' - P$ in LPN'';
2. If $\exists \alpha \vee x \mid \alpha \to x \in R_D$ or $\alpha \vee x \mid \emptyset \in R_D$, ${}^\bullet\beta \in P'' \cap P$ in LPN'';

Proof. Because $\alpha \to \beta \mid \emptyset \in R_D$, $\exists \alpha \to \beta \in R_L$, and there is no order relation between α and β in R_M.

1. If $\exists \beta \vee x \mid \emptyset \in R_D$ or $\beta \vee x \mid x \to \beta \in R_D$, then $\beta \vee x \in R_L$. $\exists \alpha \in A$ such that $\alpha = {}^\blacklozenge\beta \cap {}^\blacklozenge x$ in L, and $\exists \alpha \in T$ such that $\alpha = {}^\bullet({}^\bullet x)$ in PM. After Step 5 of Algorithm 4, ${}^\bullet\beta = \emptyset$. Because $\beta \vee x$, ${}^\bullet\beta \cup {}^\bullet x = \alpha^\bullet$ in LPN''. Then a new place p_m ($p_m = {}^\bullet\beta$) is added between α and β. Therefore, ${}^\bullet\beta \in P'' - P$ in LPN'';
2. If $\exists \alpha \vee x \mid \alpha \to x \in R_D$ or $\alpha \vee x \mid \emptyset \in R_D$, then $\alpha \vee x \in R_L$. $\exists \beta \in A$: $\beta = \alpha^\blacklozenge \cap x^\blacklozenge$ in L, and $\exists \beta \in T$: $b = (x^\bullet)^\bullet$ in PM. From Algorithm 4, $\alpha^\bullet \cup x^\bullet = {}^\bullet\beta$ in LPN''. Because $\alpha^\bullet(x^\bullet) \in P$ and $\alpha^\bullet(x^\bullet) \in P''$, we have $\beta^\bullet \in P$ and $\beta^\bullet \in P''$. Therefore, $\beta^\bullet \in P'' \cap P$ in LPN''.

□

We illustrate Algorithm 4 with an example. For the process model PM_2 in Figure 6. The event log L_2 has five traces: $L_2 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\} = \{\langle a, b, c, d, e\rangle, \langle a, b, c, e\rangle, \langle a, d, e\rangle, \langle a, d, b, c, e\rangle, \langle a, b, d, c, e\rangle\}$. The activities in trace σ_1 can be executed sequentially in PM_2. Other traces cannot be replayed accurately in PM_2. According to Algorithm 4, we can obtain the repaired LPN-based model that is shown in Figure 7. The repaired models based on the methods in [11, 12] are in Figures 8 and 9, respectively. All traces in L_2 can be replayed in three repaired models. It is clear that ours is the most concise.

Algorithm 4: Sequential Structures-based Model Repair Algorithm

Input: A process model PM and an event log L
Output: The repaired model $LPN'' = (P'', T''; F'', I'', O'', M'')$

$LPN'' = PM$
Executing Algorithm 1 to compute R_L and R_M;
Executing Algorithm 2 to compute R_D;
for $\forall \alpha \vee \beta \mid \alpha \to \beta \in R_D$ do
  $F'' = F'' - \{\alpha^\bullet \to \beta\}$;
end for
for $\forall \alpha \to \beta \mid \emptyset \in R_D$ do
  if $((\exists \beta \vee x \mid \emptyset \in R_D$ or $\beta \vee x | x \to \beta \in R_D)$ and $\alpha = {}^\blacklozenge\beta \cap {}^\blacklozenge x$ and $\alpha = {}^\bullet({}^\bullet x))$, then
    $P'' = P'' \cup \{p_m\}$ and $p_m = {}^\bullet\beta$;
    $F'' = F'' \cup \{\alpha \to p_m, p_m \to \beta\}$;
    $O'' = O'' \cup \{O''(\alpha) = p_m \vee {}^\bullet x\}$;
  end if
  else if$((\exists \alpha \vee x \mid \alpha \to x \in R_D$ or $\alpha \vee x | \emptyset \in R_D)$ and $\beta = \alpha^\blacklozenge \cap x^\blacklozenge$ and $\beta = (x^\bullet)^\bullet)$, then
    $F'' = F'' \cup \{\alpha^\bullet \to \beta\}$;
    $I'' = I'' \cup \{I''(\beta) = \alpha^\bullet \vee x^\bullet$;
  end if
  else
    continue;
  end for
return LPN''

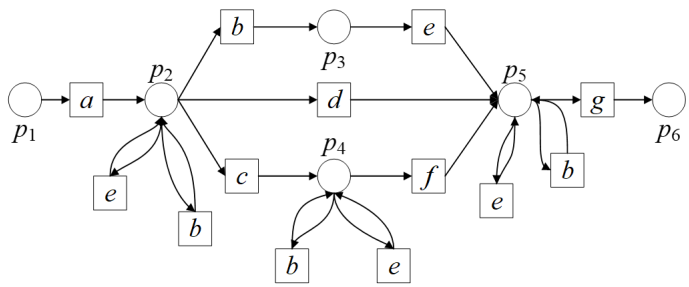

Figure 5. The repaired model based on the method [12]

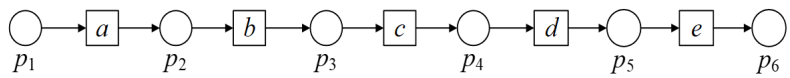

Figure 6. A Process model PM_2

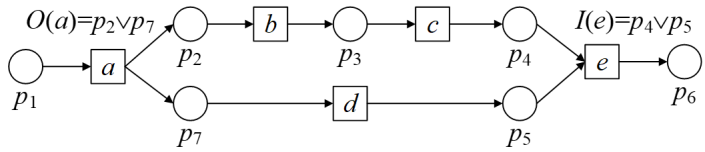

Figure 7. The repaired model based on our approach

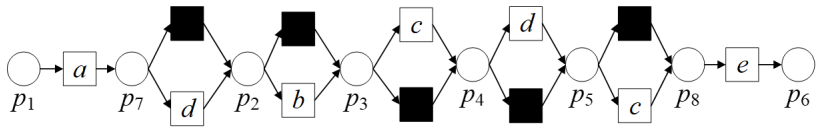

Figure 8. The repaired model based on the method [11]

5 SIMULATION EXPERIMENTS

In this section, we conduct experiments and comparative analysis. Existing approaches can repair different types of models, e.g., Petri nets [11, 12, 13, 14], LPNs [15, 16] and BPMN. The most recent model repair methods (e.g., [13, 14, 15]) focus on special structures and cases. Yet it is inappropriate to compare them with our approach. Thus, we compare our method with the most representative model repair methods in [11, 12]. The one in [11] is implemented in ProM 6.6, which is accessible at `http://www.promtools.org/prom6/`, and the one in [12] is accessible at `https://svn.win.tue.nl/repos/prom/Packages/SelectivePRepair`. Besides, the issues that the B-method [16] focuses on are somewhat related to those that our method focuses on, and we also regard it as one of the benchmark methods.

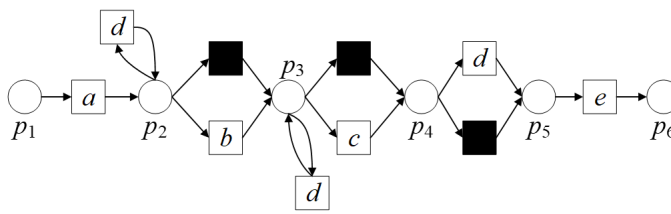

Figure 9. The repaired model based on the method [12]

5.1 Experimental Data and Original Model

We take a treatment process for tumor patients as a case study. The corresponding process model in Figure 10 is obtained based on the event log which is from a hospital in Qingdao, China. The following description is about a tumor patient's medical process. First, a patient can get an appointment with a doctor network or other communication methods before s/he goes to the hospital. And s/he can obtain a reservation number. The patient can also go directly to the hospital for registration. According to the reservation (or registration) number, the patient is called in order. Then, the doctor asks about the patient's symptoms. After questioning the patient's conditions, the doctor tells him or her the types of examinations to take. There are mainly three types of examinations: blood routine, biochemical full set, and NMR (nuclear magnetic resonance). During the diagnosis, the doctor develops the corresponding treatment plans according to the examination results and patient's conditions. There are two types of a patient's conditions:

1. if the patient is not seriously ill, s/he can receive treatment at a medical ward with medication (e.g., injecting drugs) according to the doctor's prescription. The patient can be discharged from the hospital with a period of treatment observation;
2. if the patient is seriously ill, s/he needs to undergo a surgery. At this time, the patient is admitted to the hospital and the doctor makes a diet plan for him or her.

Before the surgery, the doctor performs a preoperative evaluation. The patient does ECG and laboratory testing in turn. According to the examination results, the doctor makes a detailed surgery plan for the patient. After the surgery, s/he can be discharged from the hospital.

At first, for event logs recorded in the medical system, the process model depicted in Figure 10 can replay them correctly. However, with the change of an actual medical treatment process, some new event logs are generated. The original process model does not rightly fit these event logs. It means that the corresponding model cannot represent some activities' order relations in event logs accurately, and some structures in the model need to be changed. For instance, it can be found that patients have more choices for examination. They can do a blood routine and NMR, or a full set of NMR and biochemical tests. In addition, an ECG can be performed after laboratory testing, or only one of the ECG and laboratory testing is done for patients.

To repair the model, we collect 20 event logs (L_1-L_{20}) through a hospital's information system. All the changes mentioned above are contained in these event logs. The event logs that deviate from the original model seriously are removed manually because the model repair following these event logs is meaningless. Table 1 gives these event logs' primary attributes. Those are the number of traces, the number of activities, the number of events, and the length of traces. From Table 1,

we can find that the minimum number of traces is 117, and the maximum number of traces is 3 215. These event logs are available from `https://pan.baidu.com/s/1gxsHPVjaFOxqzqKSFcrcRw`.

5.2 Different Methods-Based Model Repair Experiments

In this sub-section, four methods, i.e., F-method [11], G-method [12], B-method [16], and our method, are adopted to repair the original process model in Figure 10. From Table 1, the largest number of traces is recorded in event log L_{20}, and it may contain the most all-sided cases. Therefore, L_{20} is regarded as experimental data to repair the process model in Figure 10. The models repaired by these three methods are respectively depicted in Figures 11, 12, 13, and 14.

The repaired model based on F-method is depicted in Figure 11. To replay all traces in L_{20}, this method represents logical concurrent relations by adding invisible transitions and self-loops. However, this causes some transitions to fires multiple times. For example, transitions "NMR", "Biochemical full set", and "Blood routine" can be executed multiple times after adding an invisible transition. Besides, four self-loops about "ECG" in the model can also be executed multiple times. The repaired model based on G-method is shown in Figure 12. Although this method can replay all traces in L_{20}, it adds single transition loop or skips a transition to the model, and also causes some transitions (e.g, "NMR" and "Laboratory test") to be executed multiple times. Figure 13 shows the repaired model based on B-method [16]. However, this method cannot replay all traces in L_{20} because it cannot represent logical concurrent relations well. Figure 14 depicts the repaired model based on our approach. From this figure, we can find that the repaired model only adds three places and four arcs to represent logical concurrent relations, and there is no loop or repetitive transition.

5.3 Model Evaluation

After repairing the model, this sub-section conducts comparison and analysis between the proposed method and three existing methods, i.e., F-method [11] and G-method [12], and B-method [16]. According to conformance checking criterion, we conduct a comparative analysis in three aspects: fitness, simplicity, and precision. Different amounts of event logs as shown in Table 1 are used to compute the fitness and precision of each repaired model. Note that, the modeling tools for LPNs have been developing. Here, we calculate values of precision and fitness of the repaired model by our method based on and B-method [16] the formulas proposed in [10].

The comparisons about fitness among them are depicted in Figure 15. Fitness is considered to be the foremost indicator to estimate a model's quality. The model's fitness value is 1 meaning that it can fit an event log completely. According to Figure 15, it is clear that the fitness values of F-method [11], G-method [12], and our method are always 1 under different event logs, i.e., the event logs L_1-L_{20} can

be accurately replayed by the three repaired models. Since B-method [16] cannot replay these 20 event logs correctly, its fitness value is always less than 1.

Event Logs	Traces	Activities	Events	Length
L_1	117	22	1 667	9–16
L_2	233	22	3 390	9–16
L_3	319	22	4 728	9–16
L_4	421	22	6 196	9–16
L_5	516	22	7 373	9–16
L_6	615	22	9 138	9–16
L_7	734	22	10 716	9–16
L_8	825	22	11 991	9–16
L_9	906	22	13 308	9–16
L_{10}	1 138	22	16 473	9–16
L_{11}	1 396	22	20 306	9–16
L_{12}	1 527	22	22 348	9–16
L_{13}	1 741	22	25 650	9–16
L_{14}	1 913	22	28 189	9–16
L_{15}	2 138	22	31 444	9–16
L_{16}	2 347	22	34 488	9–16
L_{17}	2 566	22	37 497	9–16
L_{18}	2 708	22	39 452	9–16
L_{19}	3 092	22	45 139	9–16
L_{20}	3 215	22	47 118	9–16

Table 1. The detailed information about 20 event logs

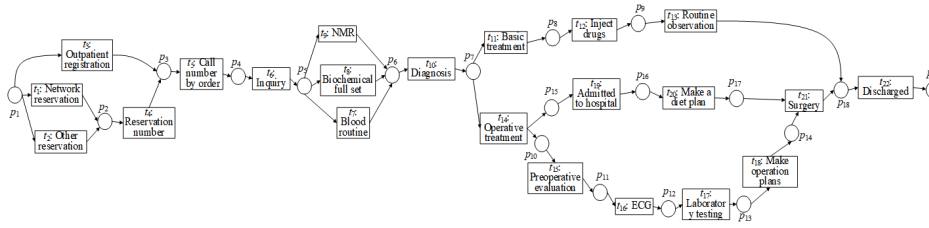

Figure 10. The original process model of tumor patients' diagnosis and treatment

Figure 16 shows the comparisons about precision among them. The higher the precision value of a model, the fewer traces are generated by the model in addition to the given traces in event logs. From Figure 16, it can be found that the repaired model based on the proposed approach has higher precision values than the repaired models based on other three approaches. The appearance of self-loops, repetitive transitions, invisible transitions, and inappropriate logical relations in the models from the other methods reduces their precision values.

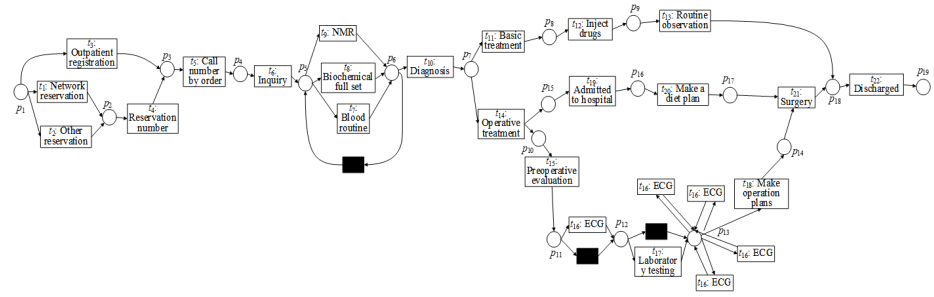

Figure 11. The repaired model based on the F-method in [11]

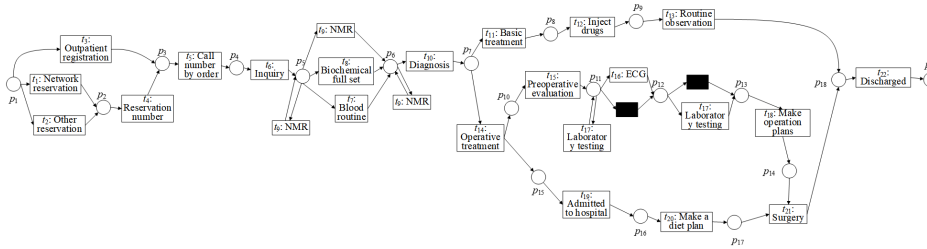

Figure 12. The repaired model based on the G-method in [12]

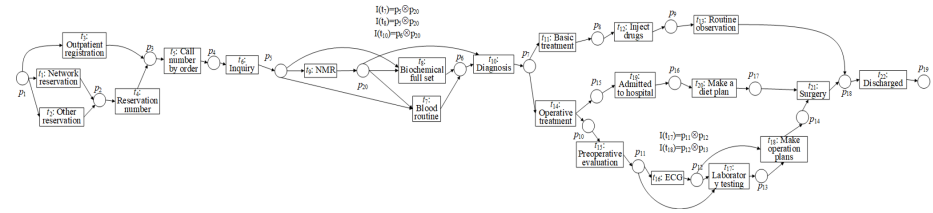

Figure 13. The repaired model based on the B-method in [16]

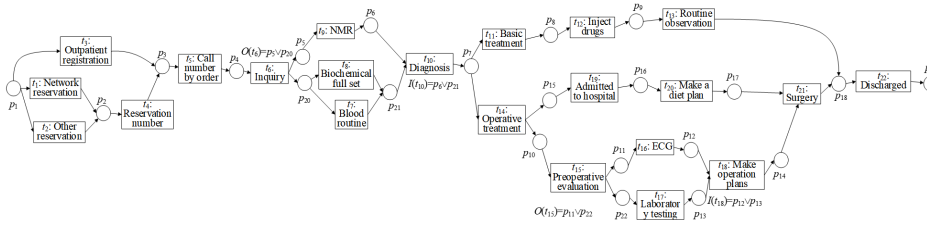

Figure 14. The repaired model based on our method

Simplicity requires that a model which is able to replay event logs should be as simple as possible. The simplicities of four repaired models are compared based on the following criteria: the number of added places ($|P|$), added arcs ($|F|$), added invisible transitions ($|\tau|$), and added transitions ($|T|$). Comparing the models in Figures 11, 12, 13, and 14 with the original model in Figure 10, we can get the following results from Table 2: the model repaired by the F-method adds four transitions, fourteen arcs, and three invisible transitions; the model repaired by G-method adds three repetitive transitions, two invisible transitions, and ten arcs; the model repaired by B-method adds one place and five arcs, but it cannot replay each event log correctly; and the model repaired based on the proposed method adds three places and four arcs only.

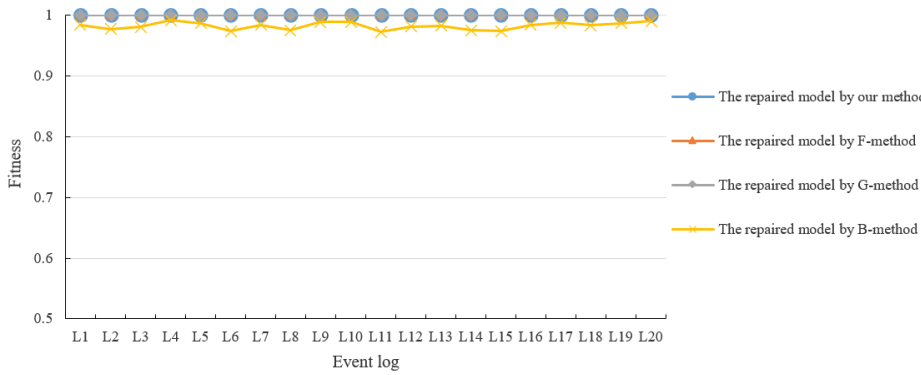

Figure 15. The comparisons about fitness between the event logs L_1–L_{20} and three different methods

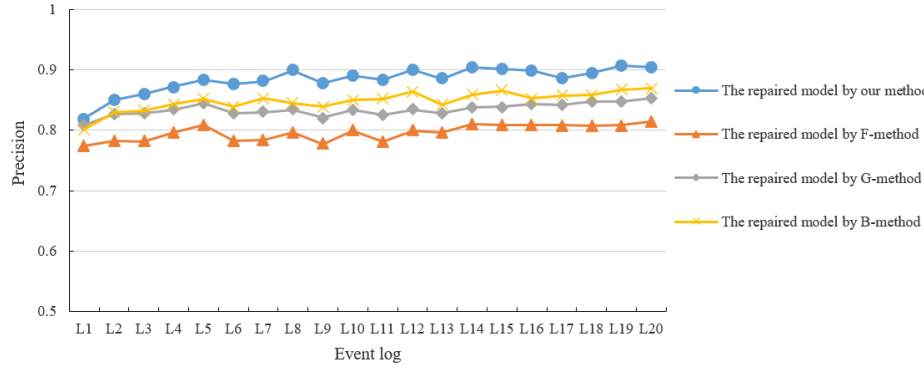

Figure 16. The comparisons about precision between the event logs L_1–L_{20} and three different methods

Model	Added $\|P\|$	Added $\|F\|$	Added $\|\tau\|$	Added $\|T\|$
The repaired model based on our method	3	4	0	0
The repaired model based on F-method	0	14	3	4
The repaired model based on G-method	0	10	2	3
The repaired model based on B-method	1	5	0	0

Table 2. The comparisons about simplicity for three different methods

With the business process changing or being updated, some activities in event logs now can be executed concurrently, but not in the past. Then we should change some of the model's structures to concurrent ones. The baselines mentioned above cannot fit event logs by changing some structures in the model, and they usually insert invisible transitions, self-loops, or inappropriate logical relations. The proposed method can deal with these situations by changing structures in the model and adding suitable logical functions. The logical functions of transitions can improve the model's precision and simplicity, and describe the actual business process more accurately. Besides, LPN-based models have been applied in many fields, such as e-commerce and medical treatment [17].

6 CONCLUSIONS

This work presents an LPN-based model repair approach to describe and represent a changing business process. Our purpose is to the change model's structures based on event logs. Some activities have concurrent relations in event logs, but cannot be executed concurrently in the model. This study designs methods to change choice or sequential structures to concurrent ones. First, a deviation set is constructed to collect the differences between a model and an event log. Then the model is repaired according to the deviation set. By adding logical functions to the model, the repaired model can accurately depict a real-life business process. Through a case study, we compare our method with other state-of-the-art methods. The results show that the model repaired by the proposed method has higher precision and a smaller net size than its peers. The proposed approach mainly considers changing models' structures to describe and represent the changing business processes accurately. As future work, we plan to propose some model repair approaches to solve more complex situations by LPNs. Other monitoring techniques such as predictive business process [35] should be taken into account.

Acknowledgement

This work was supported in part by the Natural Science Foundation of China under Grant No. 61903229, in part by the Natural Science Foundation of Shandong Province under Grant No. ZR2021MF117, and in part by the Education Ministry Humanities and Social Science Research Youth Fund Project of China under Grant No. 21YJCZH150.

REFERENCES

[1] Su, Y.—Qi, L.—Zhou, M. C.: A Backward Algorithm to Determine the Existence of Legal Firing Sequences in Ordinary Petri Nets. IEEE Robotics and Automation Letters, Vol. 8, 2023, No. 6, pp. 3190–3197, doi: 10.1109/LRA.2023.3246384.

[2] Yu, W.—Yan, C.—Ding, Z.—Jiang, C.—Zhou, M.: Analyzing E-Commerce Business Process Nets via Incidence Matrix and Reduction. IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 48, 2018, No. 1, pp. 130–141, doi: 10.1109/TSMC.2016.2598287.

[3] Garcia-Uribe, C.—López-Mellado, E.: Building Hierarchical Workflow Nets for Discrete-Event Processes Discovery. 2023 20th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), IEEE, 2023, pp. 1–7, doi: 10.1109/CCE60043.2023.10332857.

[4] van der Aa, H.—Leopold, H.—Reijers, H. A.: Efficient Process Conformance Checking on the Basis of Uncertain Event-to-Activity Mappings. IEEE Transactions on Knowledge and Data Engineering, Vol. 32, 2020, No. 5, pp. 927–940, doi: 10.1109/TKDE.2019.2897557.

[5] Brzychczy, E.—Żuber, A.—van der Aalst, W.: Process Mining of Mining Processes: Analyzing Longwall Coal Excavation Using Event Data. IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 54, 2024, No. 5, pp. 2723–2734, doi: 10.1109/TSMC.2023.3348496.

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

[8] Wen, L.—van der Aalst, W. M. P.—Wang, J.—Sun, J.: 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.

[9] Weber, P.—Bordbar, B.—Tino, P.: A Framework for the Analysis of Process Mining Algorithms. IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 43, 2013, No. 2, pp. 303–317, doi: 10.1109/TSMCA.2012.2195169.

[10] Adriansyah, A.: Alignment Observed and Modeled Behavior. Ph.D. Thesis. Eindhoven University of Technology, Eindhoven, Netherlands, 2014, doi: 10.6100/IR770080.

[11] 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.

[12] 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 (TOSEM), Vol. 25, 2016, No. 4, Art. No. 28, doi: 10.1145/2980764.

[13] Sun, Y.—Du, Y.—Li, M.: A Repair of Workflow Models Based on Mirroring Matrices. International Journal of Parallel Programming, Vol. 45, 2017, No. 4, pp. 1001–1020, doi: 10.1007/s10766-016-0438-1.

[14] Qi, H.—Du, Y.—Qi, L.—Wang, L.: An Approach to Repair Petri Net-Based Process Models with Choice Structures. Enterprise Information Systems, Vol. 12, 2018, No. 8–9, pp. 1149–1179, doi: 10.1080/17517575.2018.1432768.

[15] Xu, Y.—Du, Y.—Qi, L.—Luan, W.—Wang, L.: A Logic Petri Net-Based Model Repair Approach by Constructing Choice Bridges. IEEE Access, Vol. 7, 2019, pp. 18531–18545, doi: 10.1109/ACCESS.2019.2896079.

[16] Bai, E.—Su, N.—Liang, Y.—Qi, L.—Du, Y.: Method for Repairing Process Models with Selection Structures Based on Token Replay. Computing and Informatics, Vol. 40, 2021, No. 2, pp. 446–468, doi: 10.31577/cai_2021_2_446.

[17] Du, Y.—Qi, L.—Zhou, M.: Analysis and Application of Logical Petri Nets to E-Commerce Systems. IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 44, 2014, No. 4, pp. 468–481, doi: 10.1109/TSMC.2013.2277696.

[18] Du, Y.—Zhu, H.—Wang, L.—Liu, W.: A Method of Process Mining Based on Logic Petri Nets. Acta Electronica Sinica, Vol. 44, 2016, No. 11, pp. 2743–2751, doi: 10.3969/j.issn.0372-2112.2016.11.025 (in Chinese).

[19] Hu, Q.—Du, Y.—Yu, S.: Service Net Algebra Based on Logical Petri Nets. Information Sciences, Vol. 268, 2014, pp. 271–289, doi: 10.1016/j.ins.2013.10.014.

[20] Qi, L.—Zhou, M.—Luan, W.: A Two-Level Traffic Light Control Strategy for Preventing Incident-Based Urban Traffic Congestion. IEEE Transactions on Intelligent Transportation Systems, Vol. 19, 2018, No. 1, pp. 13–24, doi: 10.1109/TITS.2016.2625324.

[21] Qi, L.—Su, Y.—Zhou, M. C.—Abusorrah, A.: A State-Equation-Based Backward Approach to a Legal Firing Sequence Existence Problem in Petri Nets. IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 53, 2023, No. 8, pp. 4968–4979, doi: 10.1109/TSMC.2023.3241101.

[22] Xiang, D.—Liu, G.—Yan, C.—Jiang, C.: Detecting Data-Flow Errors Based on Petri Nets with Data Operations. IEEE/CAA Journal of Automatica Sinica, Vol. 5, 2018, No. 1, pp. 251–260, doi: 10.1109/JAS.2017.7510766.

[23] Jiang, Z.—Li, Z.—Wu, N.—Zhou, M.: A Petri Net Approach to Fault Diagnosis and Restoration for Power Transmission Systems to Avoid the Output Interruption of Substations. IEEE Systems Journal, Vol. 12, 2017, No. 3, pp. 2566–2576, doi: 10.1109/JSYST.2017.2682185.

[24] Yang, F.—Wu, N.—Qiao, Y.—Su, R.: Polynomial Approach to Optimal One-Wafer Cyclic Scheduling of Treelike Hybrid Multi-Cluster Tools via Petri Nets.

IEEE/CAA Journal of Automatica Sinica, Vol. 5, 2018, No. 1, pp. 270–280, doi: 10.1109/JAS.2017.7510772.

[25] You, D.—Wang, S.—Zhou, M.: Computation of Strict Minimal Siphons in a Class of Petri Nets Based on Problem Decomposition. Information Sciences, Vol. 409–410, 2017, pp. 87–100, doi: 10.1016/j.ins.2017.05.011.

[26] Wang, L.—Du, Y.—Qi, M.—Qi, H.—He, Z.: Petri Net-Based Deviation Detection Between a Process Model with Loop Semantics and Event Logs. Concurrency and Computation: Practice and Experience, Vol. 30, 2018, No. 23, Art. No. e4419, doi: 10.1002/cpe.4419.

[27] Wang, S.—You, D.—Seatzu, C.: A Novel Approach for Constraint Transformation in Petri Nets with Uncontrollable Transitions. IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 48, 2018, No. 8, pp. 1403–1410, doi: 10.1109/TSMC.2017.2665479.

[28] Yang, F.—Wu, N.—Gao, K.—Zhang, C.—Zhu, Y.—Su, R.—Qiao, Y.: Efficient Approach to Cyclic Scheduling of Single-Arm Cluster Tools with Chamber Cleaning Operations and Wafer Residency Time Constraint. IEEE Transactions on Semiconductor Manufacturing, Vol. 31, 2018, No. 2, pp. 196–205, doi: 10.1109/TSM.2018.2811125.

[29] Ma, Z.—Li, Z.—Giua, A.: Characterization of Admissible Marking Sets in Petri Nets with Conflicts and Synchronizations. IEEE Transactions on Automatic Control, Vol. 62, 2017, No. 3, pp. 1329–1341, doi: 10.1109/TAC.2016.2585647.

[30] Khaosanoi, L.—Limpiyakorn, Y.: Conformance Checking and Discovery of Information Service Request Process. 2021 14th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), IEEE, 2021, pp. 1–5, doi: 10.1109/CISP-BMEI53629.2021.9624359.

[31] Leemans, S. J. J.—Fahland, D.—van der Aalst, W. M. P.: Scalable Process Discovery and Conformance Checking. Software and Systems Modeling, Vol. 17, 2018, No. 2, pp. 599–631, doi: 10.1007/s10270-016-0545-x.

[32] Leemans, S. J. J.—Fahland, D.—van der Aalst, W. M. P.: Scalable Process Discovery with Guarantees. In: Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q. (Eds.): Enterprise, Business-Process and Information Systems Modeling (BPMDS 2015, EMMSAD 2015). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 214, 2015, pp. 85–101, doi: 10.1007/978-3-319-19237-6_6.

[33] Wang, Y.—Du, Y.: Conformance Detection Based on Extended Footprint Matrix. Journal of Shandong University of Science and Technology (Natural Science Edition), Vol. 37, 2018, No. 2, pp. 9–15, doi: 10.16452/j.cnki.sdkjzk.2018.02.002 (in Chinese).

[34] García-Bañuelos, L.—van Beest, N. R. T. P.—Dumas, M.—La Rosa, M.—Mertens, W.: Complete and Interpretable Conformance Checking of Business Processes. IEEE Transactions on Software Engineering, Vol. 44, 2018, No. 3, pp. 262–290, doi: 10.1109/TSE.2017.2668418.

[35] Tsiri, T. P.—Daniyan, I. A.—Mpofu, K.: Development of a Business Process Modelling Framework for Continuous Improvements in Organisations. 2022 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), 2022, pp. 675–680, doi: 10.1109/IEEM55944.2022.9989810.

Yuyue Du received his 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 Computer, Shandong Xiehe University, Jinan, China, and Professor of computer science and technology at the 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, Web services, and workflows.

Yuhua Xu received her Ph.D. degree from the College of Electronic and Information Engineering, Tongji University, Shanghai, China, in 2024. She is currently a lecturer at the School of Information and Intelligent Science, Donghua University. Her research interests include graph neural networks and graph representation learning.

Liang Qi received his Ph.D. degree in computer software and theory from the Tongji University, Shanghai, China in 2017. From 2015 to 2017, he was a Visiting Doctoral Student at the New Jersey Institute of Technology. He is currently Associate Professor in the Shandong University of Science and Technology. He has published 100+ papers in journals and conference proceedings (25 in IEEE Transactions). He received the Best Student Paper Award-Finalist in the 15th IEEE International Conference on Networking, Sensing and Control (ICNSC 2018) and the Best Paper Award-Finalist in the 3rd IEEE International Conference on Automation in Manufacturing, Transportation and Logistics (iCaMaL2023). His current research interests include manufacturing systems, Petri nets, optimization, machine learning, and intelligent transportation.

Yijia Liu is now pursuing her B.Sc. degree in the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China. Her current research interests lie in optimization theory and application.

Qun Wang holds her Master of Engineering Management from the Ocean University of China. She serves as Director of the Shandong Computer Society, Director of Shandong Software Industry Industry-Education Alliance, and a member of the Jinan Vocational Education Society. In 2008, she joined the Shandong Shichuang Software Training Institute of Ambow Education Group and served as the Director of the Training Center. In 2022, she joined the School of Computer Science and Technology, Shandong Xiehe University, Jinan, China, where she currently holds the position of Deputy Dean of Research. Her research interests include machine vision, neural networks, and deep learning.

Mei Chen received her B.Sc. degree in computer science and technology from the Dezhou University, Dezhou, China in 2005 and her M.Sc. degree in computer application technology from the Suzhou University, Suzhou, China in 2008. She is Associate Professor at the College of Computer, Shandong Xiehe University, Jinan, China. She has published over 20 papers in domestic and international journals and conferences, including 5 SCI and EI indexed papers. Her current research interests include digital image processing and object detection.