Received January 19, 2019, accepted January 27, 2019, date of publication January 30, 2019, date of current version February 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2896079

A Logic Petri Net-Based Model Repair Approach by Constructing Choice Bridges

YUHUA XU, YUYUE DU, LIANG QI, (Member, IEEE), WENJING LUAN, (Student Member, IEEE), AND LU WANG
College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China

Corresponding author: Yuyue Du (yydu001@163.com)

This work was supported in part by the Taishan Scholar Construction Project of Shandong Province, in part by the Key Research and Development Program of Shandong Province under Grant 2018GGX101011, and in part by the Natural Science Foundation of Shandong Province under Grant ZR2018MF001.

ABSTRACT Process models can be discovered from event logs generated by the enterprise information system. As business processes' frequently changing, some activities in event logs may belong to different choice branches, while the actual model can only replay activities on the same choice branch. Thus, the model needs to be repaired to describe the business process accurately. For activities occurring in a choice structure, although the model repaired by the existing methods has good fitness, its structure may have changed a lot. This paper proposes a new model repair method based on logic Petri nets. First, we define a new event log type—choice deviation sub-log. Then, we find deviation positions according to the principle of token replaying. Next, we add bridges among choice branches to repair the model. We conduct experiments on some cases from cancer treatment processes in a hospital. The effectiveness and correctness of our method can be illustrated. The model repaired by our method is similar to the original one, and it has a higher precision compared with the existing model repair methods.

INDEX TERMS Logic Petri net, process mining, model repair, choice structures.

I. INTRODUCTION

In recent years, various enterprise organizations have started to use information systems for business process management. As a result, there are a large number of event logs recorded in enterprise information systems. Process mining can extract knowledge from event logs recorded in modern information systems, and discover, detect, and improve the actual business processes [1]. Process mining has mainly three aspects: process discovery, consistency checking, and process enhancement [2]–[4]. Process discovery takes event logs as input data and then construct a model to describe event logs. Consistency checking is an essential criterion for evaluating the quality of models. It refers to comparing the process model with the event logs, i.e., replaying a sequence of events called a trace in the model to analyze the consistency between model behaviors and log behaviors. Existing consistency checking methods include token replaying, footprint comparison, alignment, etc. Process enhancement improves or extends process models based on event logs. Some studies repair or expand the model based on the deviations between log behaviors and model behaviors. In other studies, the unreasonable process model is repaired based on three dimensions: structural distance, behavioral distance, and scheme defects, which are minimized by a simulated annealing algorithm.

The quality of a process model is measured from mainly four metrics: fitness, precision, simplicity, and generalization [4], [5]. Among them, fitness is the most important indicator. If a model can replay all traces in event logs, then its fitness value is high. Precision requires that a process model should only replay the activities in event logs. Simplicity requires that a process model which can replay event logs is as simple as possible. Generalization means that a model should allow activities to occur in the future.

With the growth of enterprises and organizations, the existing business processes are difficult to adapt to the development of enterprises and organizations. When a current model is inconsistent with the actual execution process, a new process model can be obtained through some process discovery technologies. Alpha algorithm [6] and its derivative algorithms [7], [8] can mine a process model according to the order relations between activities. The appearance of

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiwu Li.

the derivative algorithms solve the problems that the Alpha algorithm cannot effectively mining models, which include repetitive activities, invisible transitions, and some specific structures. For example, the method in [7] can mine non-free choice structures, and invisible transitions can be solved by [8]. Although these technologies can discover a new process model, they are often difficult to ensure that the new model is similar to the original one.

For a process model that cannot reproduce some activities in event logs, a better approach is to repair the original model [9], [10]. The repaired model can replay most of the event logs and retain the advantages of the original model as much as possible. For example, Fahland and Aalst [9] propose a model repair method. This method first finds the maximal sequences of log moves according to the alignment between event logs and the process model. Each maximal sequence of log moves that occur at the same location is a non-fitting sub-trace. These sub-traces which are at the same location construct a non-fitting sub-log. Then a loop is discovered and can replay the sub-log, or a corresponding sub-process can be discovered and is then added to the original model as a self-loop. For moves on model in the optimal alignment, we can add invisible transitions to the original model. Goldratt's method [10] first identifies legal moves, which can contribute to the cost of optimal alignment between an event log and a model. Then an order pair set is produced according to the cost function of these moves and constraints. Finally, it adds a single transition as a self-loop or invisible transition to the model. The model repaired by the above methods has high fitness, and most of the traces in event logs can be replayed by the repaired model. In addition, other recently model repair methods focus on repairing a particular structure. For example, the method proposed in [11] computes choice recognition pairs at first. Next, the order relations between transitions are used to find model repair positions in choice structures. Finally, it repairs the model at these positions. The method proposed in [12] can add new activities in choice structures. It introduces a new deviation type — choice branch deviation. If we find this deviation in a choice structure, we can add this deviation as a branch to the original model.

When some activities in an event log belong to different choice branches and the corresponding model can only replay activities which belong to the same branch, we need to repair the model. The existing methods repair the model by adding invisible transitions or repetitive transitions as self-loops. For cases where activities in an event log belong to different choice branches, this paper proposes a new model repair method. The new method is different from the one in previous work [13], which can repair process models with logic concurrent and casual relations according to extended order relations. In a process model, the token is an intuitive representation of the dynamic characteristics of a network system. In this paper, our new model repair method is based on the principle of token replaying. The new method can add bridges among choice branches in the original model. The model repaired by our method can replay activities which belong to the same choice branch or different choice branches. In the process of model repair, we can find that some logic relations among activities. YAWL [14], workflow language for Petri nets [15], and logic Petri net [16]–[21] can accurately represent the relations among activities. As our previous work, we adopt logic Petri nets to describe repaired models. In this paper, the main contributions are as follows:

(1) We propose a new sub-log type named choice deviation sub-log. The activities in a choice deviation sub-log belong to a choice structure but belong to different choice branches.

(2) Combined with the principle of token replaying, we can find deviation positions effectively by replaying traces in the model. Then we repair choice structures in the original model by adding bridges among branches. We repair the model in a dynamic way, i.e., once the deviation position is identified, we repair and update the model. Then we continue to find deviation positions based on the new model.

(3) We conduct simulation experiments on cancer patients' diagnosis and treatment processes. The experimental results ensure the correctness and effectiveness of our method.

The rest of this paper is organized as follows. Section II reviews some basic concepts. An approach to finding choice deviation sub-logs is given in Section III. By determining whether a choice structure contain parallel structures, Section IV designs two approaches to repair the process model. Experiments are executed in Section V. Section VI concludes this work.

II. PRELIMINARIES

Petri nets [22]–[34] have unique advantages in modeling and analyzing discrete events systems. The model established by Petri nets can describe not only the structure but also the operation of the systems. This section briefly recalls some related notions and concepts of Petri nets [36]–[49], logic Petri nets [16]–[21], and process tree [35].

Definition 1 (Multiple Sets): Let S be a set. A multi-set Z over S is a function $Z\colon S \rightarrow \mathbb{N}_+$, where $\mathbb{N}_+$ represents a positive integer set, i.e., $\mathbb{N}_+ = \{1, 2, \cdots\}$. Notation $\mathcal{B}(S)$ denotes the set of all multi-sets over S.

Definition 2 (Traces and Event Logs): Let $\mathcal{A}$ be the set of all activities, and $A \subseteq \mathcal{A}$. If $\sigma \in A^*$ is a sequence of activities, and A^* denotes a set of finite sequence on A, then σ is called a trace. $\&(\sigma)$ denotes the set of all activities in σ. $L \in \mathcal{B}(A^*)$ is a multi-set over traces called an event log.

Definition 3 (Element Position): Let $v = (a_1, a_2, \cdots, a_n)$ be a tuple with n elements, where $a_i \in S$, and $1 \leq i \leq n$. $v[i] = a_i$ denotes the i-th element of v.

For example, if $v = (a, b)$, and $a, b \in S$, then $v[1] = a$ and $v[2] = b$.

Definition 4 (Pre-Set and Post-set): $N = (P, T; F)$ is a net where P is a finite place set, T is a finite transition set, and F is a set of arcs from a place to a transition or from a transition to a place. For $\forall x \in P \cup T$, we have

(1) ${}^\bullet x = \{y | y \in P \cup T \wedge (y, x) \in F\}$, ${}^\bullet x$ denotes the pre-set of x; and

(2) $x^\bullet = \{y | y \in P \cup T \wedge (x, y) \in F\}$, $x^\bullet$ denotes the post-set of x.

Definition 5 (Petri Net): $PN = (P, T; F, M)$ is a Petri net, where

(1) $N = (P, T; F)$ is a net;

(2) $M: P \rightarrow \mathbb{N}_+$ is a marking function, and for $\forall p \in P$, $M(p)$ represents the number of tokens in p; and

(3) Transitions firing rules:

a) For $t \in T$, t is enabled at a marking M, denoted by $M[t >$ if $\forall p \in {}^\bullet t$: $M(p) \geq 1$; and

b) If $M[t >$, t can be fired, and a new marking M' is generated from M, represented by $M[t > M'$, where

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

$R(M)$ denotes the set of all reachable markings.

Note that if the maximum capacity of each place is one, then *PN* is safe. In this paper, we use $P^{M_k} = \{p | M_k(p) = 1\}$ to denote the set of places which contain one token at marking M_k.

A block-structured Petri net is a Petri net that can be divided recursively into several parts, which contains four basic Petri net-based structures, i.e., sequential, parallel, choice, and loop structures [25].

Definition 6 (Process Model): $N_S = (PN, \alpha, M_i, M_f)$ is a process model, where

(1) $PN = (P, T; F, M)$ is a block-structured Petri net, where it has a unique initial place $p_i \in P$, i.e., ${}^\bullet p_i = \emptyset$, a unique final place $p_o \in P$, i.e., $p_o^\bullet = \emptyset$, and $\forall x \in P \cup T$ is on a path from p_i to p_o;

(2) $\alpha: T \rightarrow A \cup \{\tau\}$ is a mapping function for transitions to activities, and τ denotes an invisible transition; and

(3) M_i is an initial marking, where $M_i(p_i) = 1$, and others are 0. M_f is a final marking, where $M_f(p_o) = 1$, and others are 0;

(4) Transitions firing rules: M' is a new marking generated from M if a sequence $\rho \in T^*$ satisfies $M[\rho > M'$. A sequence $\rho \in T^*$ is a complete firing sequence if $M_i[\rho > M_f$.

$R(N_S, M_i)$ denotes the set of all reachable markings.

In Definition 6, a process model is represented by a block-structured Petri net. Unless specifically indicated, only visible transitions can be marked as activities.

In this paper, we only consider block-structured Petri nets. A block-structured Petri net N_{S1} is shown in Fig. 1. M_{i1} is the initial marking, where only $M_{i1}(p_1) = 1$, i.e., $P^{M_{i1}} = \{p_1\}$. M_{f1} is the final marking, where only $M_{i1}(p_9) = 1$, i.e., $P^{M_{f1}} = \{p_9\}$. $\rho_1 = <a, b, c, e, f, i>$ is a complete firing sequence because $M_{i1}[\rho_1 > M_{f1}$.

A Process tree is an abstract representation of a block-structured Petri net. We can determine the structures of a block-structured Petri net according to the corresponding process tree. The leaf nodes of a process tree represent activities

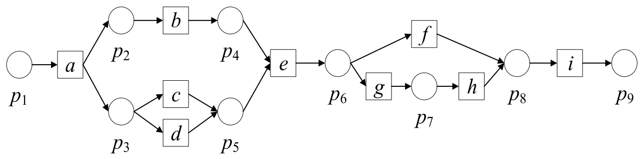

FIGURE 1. A block-structured Petri net N_{S1}.

and other nodes represent operators which describe how their children are to be combined. The definition is as follows [35].

Definition 7 (Process Tree): Let $\mathcal{A}$ be the set of all activities, and $A \subseteq \mathcal{A}$. Notation $\oplus = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ denotes an operator set, τ denotes an invisible transition, and we recursively define a process tree as follows:

(1) $a \in A \cup \{\tau\}$ is a process tree;

(2) If $PT_1 - PT_n$ are n process trees, then $\oplus(PT_1, PT_2, \cdots, PT_n)$ is also a process tree, where

a) $\rightarrow$ denotes the sequential relation among $PT_1 - PT_n$;

b) $\times$ denotes the choice relation among $PT_1 - PT_n$;

c) $\wedge$ denotes the parallel relation among $PT_1 - PT_n$;

d) $\circlearrowleft$ denotes the loop relation among $PT_1 - PT_n$, where PT_1 denotes the loop body, $PT_2 - PT_n$ denotes the loop path.

Definition 8 (Logic Petri Net): $LPN = (P, T; F, I, O, M)$ is called a logic Petri net, where

(1) P is a finite place set;

(2) $T = T_D \cup T_I \cup T_O$ is a finite transition set with $T \cap P = \emptyset$, $T \cup P \neq \emptyset$, and $\forall t \in T_I \cap T_O$: ${}^\bullet t \cap t^\bullet = \emptyset$, where

a) T_D represents a traditional transition set as defined in Definition 5;

b) T_I represents a logic input transition set, where for $\forall t \in T_I$, the input places ${}^\bullet t$ are restricted by a logic input function $f_I(t)$;

c) T_O represents a logic output transition set, where for $\forall t \in T_O$, the output places $t^\bullet$ are restricted by a logic output function $f_O(t)$;

(3) $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs;

(4) I is a mapping from a logic input transition to a logic input function, i.e., $\forall t \in T_I, I(t) = f_I(t)$;

(5) O is a mapping from a logic output transition to a logic output function, i.e., $\forall t \in T_O, O(t) = f_O(t)$;

(6) M:$P \rightarrow \{0, 1\}$ is a marking function, and for $\forall p \in P$, $M(p)$ represents the number of tokens in p; and

(7) Transition firing rules:

a) For $\forall t \in T_D$, the firing rules of t are the same as Petri nets;

b) $\forall t \in T_I$ can be fired only if $f_I(t)|_M = {}_\bullet\mathrm{T}_\bullet$, i.e., the logic input function $f_I(t)$ of t is true. If $M[t > M'$, $\forall p \in {}^\bullet t$: $M'(p) = 0$, $\forall p \in t^\bullet$: $M'(p) = 1$, and $\forall p \notin t^\bullet \cup {}^\bullet t$: $M'(p) = M(p)$; and

c) $\forall t \in T_O$ can be fired only if $f_O(t)|_M = {}_\bullet\mathrm{T}_\bullet$, i.e., the logic output function $f_O(t)$ of t is true. If $M[t > M'$, $\forall p \in {}^\bullet t$: $M'(p) = 0$, $\forall p \in t^\bullet$: $M'(p) = 1$, and $\forall p \notin t^\bullet \cup {}^\bullet t$: $M'(p) = M(p)$;

(8) If $p_1 - p_n$ $(n \geq 2)$ are pre-set (or post-set) of logic input (output) transition t:

a) $p_1 \otimes p_2 \cdots \otimes p_n$ denotes that t is enabled if only one of $p_1 - p_n$ contains a token (or t is fired and a token is generated in only one of $p_1 - p_n$);

b) $p_1 \wedge p_2 \cdots \wedge p_n$ denotes that t is enabled if each of $p_1 - p_n$ contains a token (or t is fired and a token is generated in each of $p_1 - p_n$); and

c) $p_1 \vee p_2 \cdots \vee p_n$ denotes that t is enabled if at least one of $p_1 - p_n$ contains a token (or t is fired and a token is generated in at least one of $p_1 - p_n$).

Fig. 2 gives an example of a logic Petri net denoted by LPN_1. a is a traditional transition, b is a logic input transition, c is a logic output transition, and d is a traditional transition. $I(b) = p_3 \wedge (p_2 \vee p_4)$ is the logic input function of b. It represents when b is fired, there may be three cases: (1) only p_2 and p_3 contain tokens; (2) only p_3 and p_4 contain tokens; (3) p_2, p_3, and p_4 all contain tokens. $O(c) = p_7 \otimes p_8$ represents the logic output function of c, and after c is fired, only one of p_7 and p_8 contains a token.

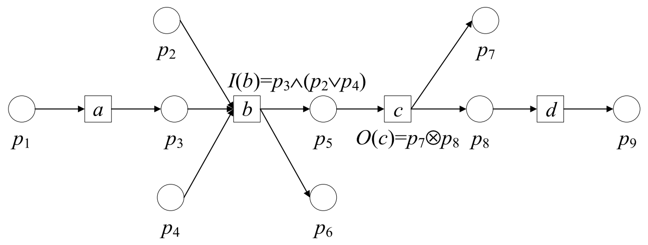

FIGURE 2. A logic Petri net model LPN_1.

III. FINDING CHOICE DEVIATION SUBLOGS

In a real process, a trace may contain some activities which belong to different choice branches, while the corresponding model can only replay activities which belong to the same choice branch. Therefore, we need to repair the choice structures of the model so that it can replay this type of trace. In this paper, we assume that all deviations occur in choice structures, and there is no new activity or repetitive activity in any trace. When the event log is not consistent with the original model, we repair the model according to the event log.

A. FINDING ALL CHOICE STRUCTURES

To repair choice structures, we need to discover choice structures at first. In this paper, we assume that each process model is a block-structured Petri net. Then each process model has a unique process tree [35], and the process tree records structures and transitions of a process model. Each operator in a process tree has a straightforward formal translation to a process model. For example, if there is a choice structure in a process model, the corresponding process tree has a node n_t ="×". By traversing subtrees of n_t, we can find all leaf nodes. For convenience, $\mathrm{L}_i(n_t)$ $(1 \le i \le m)$, $\mathrm{LL}(n_t)$, and $\mathrm{RL}(n_t)$ denote all leaf nodes of the i-th subtree, the first leaf node of the left-most subtree, and the last leaf node of the right-most subtree of n_t, respectively. According to the four basic structures of a block-structured Petri net, all choice branches start from a place p_i and end at a place p_o in a choice structure.

Definition 9 (Choice Pre-Post Pair): Let $N_S = (PN, \alpha, M_i, M_f)$, and PT be the process tree of N_S. $cp = (p_i, p_o)$ is a choice pre-post pair, where

(1) $\exists n_t =$ " × " $\in PT$; and

(2) $p_i = {}^{\bullet}\mathrm{LL}(n_t)$ and $p_o = \mathrm{RL}(n_t)^{\bullet}$.

A choice pre-post pair records the unique pre-set and post-set of a choice structure in a process model. The choice pre-post pair set S_{CP} contains all choice pre-post pairs, i.e., $S_{CP} = \{(p_i, p_o) | p_i = {}^{\bullet}\mathrm{LL}(n_t), p_o = \mathrm{RL}(n_t)^{\bullet}, \forall n_t =$ " × " $\in PT\}$.

The process tree PT_1 of N_{S1} is shown in Fig. 3, and we can get the choice pre-post pair set is $S_{CP} = \{(p_3, p_5), (p_6, p_8)\}$.

Definition 10 (Choice Branch Tuple) Let $N_S = (PN, \alpha, M_i, M_f)$, PT be the process tree of N_S, and cp be a choice pre-post pair. $ct[cp] = (t_1, t_2, \cdots, t_n)$ is a choice branch tuple, where

(1) $\exists n_t =$ " × " $\in PT$; and

(2) $t_k \in \mathrm{L}_i(n_t)$, $1 \le k \le n$, $1 \le i \le m$;

$CTS[cp]$ is a set containing all branch tuples in a choice structure, i.e., $CTS[cp] = \{(t_1, t_2, \cdots, t_n) | t_k \in \mathrm{L}_i(n_t), 1 \le k \le n, 1 \le i \le m$, and $\forall n_t =$ " × " $\in PT\}$. And $CTS = \{CTS[cp] | \forall cp \in S_{CP}\}$ denotes all choice branch tuples in a process model.

From Fig. 3, we can find the choice branch tuples of PT_1, i.e., $CTS = CTS[(p_3, p_5)] \cup CTS[(p_6, p_8)] = \{(c), (d), (f), (g, h)\}$.

$$PT_1 = \rightarrow (a, \wedge(b, \times(c, d), e, \times(f, \rightarrow (g, h)), i)$$

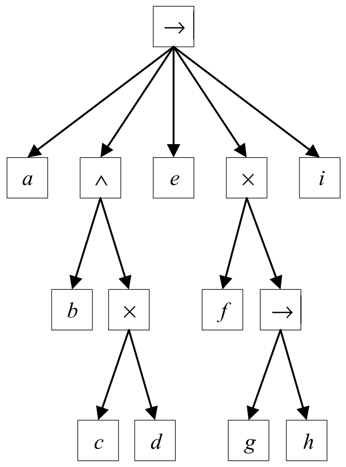

FIGURE 3. The process tree PT_1 of N_{S1}.

B. FINDING ALL CHOICE SUBLOGS WITH DEVIATIONS

To repair choice structures, we also need to collect choice sub-logs with deviations. In this paper, we only repair the models containing choice structures, and the models can only replay activities in the same choice branch. In this sub-section, we preprocess event logs for repairing model conveniently. Given an event log, a choice pre-post pair, and a choice branch tuple set, we can judge whether there is a choice sub-log with deviations. The formal definition is as follows.

Definition 11 (Projection): Let T be a set of all transitions, and $Ts \subseteq T$. For $\forall r \in Ts^*$, $r|Ts = \{t|t \in Ts$ and $t \in r\}$ is called a projection of r on Ts.

Definition 12 (Multi-Projection): Let T be a set of all transitions, and $Ts \subseteq T$. For $\forall R \in \mathcal{B}(Ts^*)$, $R \downarrow Ts = \{t|t \in Ts$ and $t \in R[i]$, $1 \leq i \leq |R|\}$ is called a multi-projection of R on Ts.

For example, let $L_1 = \{\sigma_1, \sigma_2\} = \{<b, d>, <c, e, f, g, h>\}$ and $Ts = \{a, b, c, d, e, f, g, h, i\}$, then we can get $\sigma_1|Ts = \{b, d\}$ and $L_1 \downarrow Ts = \{b, d, c, e, f, g, h\}$.

Definition 13 (Choice Deviation Sub-Trace and Choice Deviation Sub-Log): Let $\sigma \in L, S_{CP}$ be a choice pre-post pair, and CTS be a choice branch tuple set. $cl = cl[1], cl[2], \cdots, cl[|cl|]$ is called a choice deviation sub-trace, where

(1) $\exists CTS[cp] \in CTS$, s.t. $cl = \sigma|(CTS[cp] \downarrow T) \neq \emptyset$, i.e., cl is a projection of σ on $CTS[cp] \downarrow T$;

(2) $({}^\bullet cl[1], cl[scl|]^\bullet) \in S_{CP}$, i.e., $({}^\bullet cl[1], cl[|cl|]^\bullet)$ is the pre-post pair of a choice structure; and

(3) $cl \notin CTS[cp]$, i.e., cl contains activities on different branches.

If there exist multiple sets of choice deviation sub-traces $CL \in \mathcal{B}(A^*)$, then we call CL a choice deviation sub-log.

For example, let $\sigma_3 = <a, b, c, e, f, h, i>$ be a trace. According to N_{S1} and PT_1, we can get $CTS[(p_3, p_5)] \downarrow T = \{c, d\}$ and $CTS[(p_6, p_8)] \downarrow T = \{f, g, h\}$, then we have $cl_1 = \sigma_3|(CTS[(p_3, p_5)] \downarrow T) = <c>$ and $cl_2 = \sigma_3|(CTS[(p_6, p_8)] \downarrow T) = <f, h>$. Because $cl_1 \in CTS[(p_3, p_5)]$, cl_1 is not a choice deviation sub-trace. cl_2 satisfies $({}^\bullet cl_2[1], cl_2[|cl|]^\bullet) \in S_{CP}$ and $cl_2 \notin CTS[(p_3, p_5)]$, then cl_2 is a choice deviation sub-trace.

According to the given definition, we can obtain a choice deviation sub-log by the following algorithm.

Algorithm 1: Choice Deviation Sub-Log Computation

```
Input: An event log $L$, a choice pre-post pair set $S_{CP}$, and a choice branch tuple set $CTS[cp]$
Output: the choice deviation sub-log $CL$
1.  $CL \leftarrow \emptyset$;
2.  $CT = CTS[cp] \downarrow T$;
3.  for ($i = 1, \sigma_i \in L; i++; i \leq |L|$) do
4.      $\sigma_i' = \sigma_i|CT$;
5.      if $(({}^\bullet\sigma_i'[1], \sigma_i'[|\sigma_i'|]^\bullet) \in S_{CP})$, then
6.          if ($\sigma_i' \notin CTS[cp]$), then
7.              $cl = \sigma_i'$;
8.          end if
9.          else
10.             $cl = \emptyset$;
11.         $CL \leftarrow CL \cup cl$;
12.     end if
13.     else
14.         continue;
15. end for
16. return $CL$.
```

According to Algorithm 1, Step 1 initializes variable CL. Step 2 uses CT to record the multi-projection of $CTS[cp]$ on T. We can find all transitions in a choice structure according to Step 2. Steps 3-15 are used to find all choice deviation sub-traces. For $\forall\sigma \in L$, we can get a choice sub-trace σ' according to Step 4, but σ' may not have deviations. Then we need to filter the obtained choice sub-trace σ'. According to Definition 13, if the pair which consists of the pre-set of σ_i' [1] and the post-set of σ_i' $[|\sigma_i'|]$ belongs to a choice pre-post pair set S_{CP}, and the transitions in σ_i' do not belong to any choice branch, i.e., $\sigma_i' \notin CTS$, then we can get a choice deviation sub-trace cl. Otherwise, cl is empty. Finally, we can get a choice deviation sub-log CL by Steps 11-16.

After collecting all choice deviation sub-logs, we next repair the process model to replay these sub-logs.

IV. MODEL REPAIR CONTAINING CHOICE STRUCTURES

For a process model that contains a choice structure, any branches in the choice structure can occur. In the actual process, the following situation may exist: after some activities in one choice branch are executed, the activities in another branch are executed. At this time, we can find that the choice activities contained in a trace do not belong to the same choice branch tuple, i.e., a trace contains activities in different choice branches. However, the original process model cannot reproduce this type of trace. In this section, we repair the process models containing choice structures according to choice deviation sub-logs. To clearly express logic relations among transitions, the repaired models are denoted by logic Petri nets. We find deviation positions by comparing the pre-set (post-set) of transitions with P^{M_k} (cf. Definition 5). If a choice structure contains parallel elements, the pre-set (post-set) of a transition may be multiple. Therefore, we need to classify sub-logs by finding parallel elements at first.

Definition 14 (Parallel Recognition Pair): Let $N_S = (PN, \alpha, M_i, M_f)$, and PT be the process tree of N_S. $rt = (t_i, t_o)$ is a parallel recognition pair, where

(1) $\exists n_t =$ " $\wedge$ " $\in PT$; and

(2) $t_i = {}^\bullet({}^\bullet\mathrm{LL}(n_t)), t_o = (\mathrm{RL}(n_t)^\bullet)^\bullet$.

The parallel recognition pair set S_{RT} contains all parallel pre-post pairs, i.e., $S_{RT} = \{(t_i, t_o)|t_i = {}^\bullet({}^\bullet\mathrm{LL}(n_t)), t_o = (\mathrm{RL}(n_t)^\bullet)^\bullet, \forall n_t =$ " $\wedge$ " $\in PT\}$.

Definition 15 (Parallel Transition Pair): Let $N_S = (PN, \alpha, M_i, M_f)$, and PT be the process tree of N_S. $pt = (t_1, t_2)$ is a parallel transition pair, where

(1) $\exists n_t =$ " $\wedge$ " $\in PT$; and

(2) $t_1 \in \mathrm{L}_i(n_t), t_2 \in \mathrm{L}_j(n_t)$, and $i \neq j$.

S_{PT} is a set containing all parallel transition pairs, i.e., $S_{PT} = \{(t_1, t_2,) |t_1 \in \mathrm{L}_i(n_t), t_2 \in \mathrm{L}_j(n_t), i \neq j$ and $\forall n_t =$ " $\wedge$ " $\in PT\}$.

In this paper, we use $S_{RT} \downarrow T \cup S_{PT} \downarrow T$ to denote all parallel elements in a process model.

The concepts are used to judge whether a choice deviation sub-log contains parallel elements. If $cl|(S_{RT} \downarrow T \cup S_{PT} \downarrow T) = \emptyset$, that means no parallel activities in cl. Otherwise, cl contains parallel activities.

A. MODEL REPAIR WITHOUT PARALLEL STRUCTURES

In this sub-section, according to choice deviation sub-logs, we repair the process model containing choice structures, and there are no parallel structures in choice structures. We find deviations according to the principle of token replaying. Once the deviation position is found, then we repair the model, rather than repair the model after all deviation positions are found.

Theorem 1: Let cl be a choice deviation sub-trace. There exists a deviation if $cl|(S_{RT} \downarrow T \cup S_{PT} \downarrow T) = \emptyset$ and ${}^\bullet cl[k] \notin P^{M_{k-1}}$.

Proof: According to given definitions, $S_{RT} \downarrow T \cup S_{PT} \downarrow T$ is a set which contains all parallel elements in a model. $cl|(S_{RT} \downarrow T \cup S_{PT} \downarrow T) = \emptyset$ denotes no parallel element in cl. According to $cl[1]$ and $cl[|cl|]$, we can determine a choice structure which cl belongs to, and the choice structure does not contain a parallel structure. After $cl[k-1]$ is fired, $P^{M_{k-1}}$ denotes a set of places containing one token at marking M_{k-1}. If ${}^\bullet cl[k] \notin P^{M_{k-1}}$, it means that there is no token in the pre-set of $cl[k]$. Then $cl[k]$ cannot be fired in the model. Here, a deviation exists. ■

According to Theorem 1, we give the following algorithm to repair the choice structures without parallel structures.

Algorithm 2 first finds choice deviation sub-traces which do not contain parallel elements (cf. Steps 2-3). Then we fire transitions sequentially according to the ordering of events in the trace. According to Definition 13, a choice deviation sub-trace contains at least one first transition of a choice branch. Then we can fire cl [1], and get P^{M_1} (a set of places which contain one token at M_1 after cl [1] is fired). Next, we compare the pre-sets of other activities in cl with P^{M_k} to identify deviation positions, and determine which transitions need to be modified as logic transitions. For activities in cl, we deal with them in the following two cases: (1) If the pre-set of $cl[k]$ does not belong to $P^{M_{k-1}}$, i.e., ${}^\bullet cl[k] \notin P^{M_{k-1}}$, then $cl[k]$ is regarded as a logic input transition (cf. Step 6). Notation OP_l represents the pre-set of $cl[k]$, and NP_l represents the set of places containing one token after $cl[k$-1] is fired (cf. Steps 7-8). Then we add arcs from NP_l to $cl[k]$, and obtain the input function of $cl[k]$: I' $(cl[k]) = OP_l \otimes NP_l$ according to [20] (cf. Steps 9-10). Then we can fire $cl[k]$ according to $P^{M_{k-1}}$ and I' $(cl[k])$, and update P^{M_k} (cf. Steps 11-14). (2) If the activities in cl do not belong to the above case, then we execute Steps 15-18 directly. According to Steps 15-18, $cl[k]$ can be fired according to $P^{M_{k-1}}$. We have $P^{M'_k} = cl[k]^\bullet$, and then we update the P^{M_k}, i.e., $P^{M_k} = P^{M'_k}$. Finally, we can obtain the repaired logic Petri net-based model when all activities in cl are fired.

Now, we use an example to illustrate the steps of Algorithm 2. Assuming that $L_2 = \{\sigma_4\} = \{<a, d, f, i, j, l>\}$, a process model N_{S2} $(M_{i2}(p_1) = 1)$ is shown in Fig. 4. According to the above definitions and Algorithm 1, we can see that (p_2, p_9) is one of choice pre-post pairs, and $cl_3 = (d, f, i, j)$ is a choice deviation sub-trace. Choice activities d, f, i, and j belong to different choice branches in N_{S2}. According to Algorithm 2, d is the first activity of cl. According to the principle of token replaying, after a is fired, we can get $M_{i2}(p_2) = 1$, i.e., $P^{M_{i2}} = \{p_2\}$. Then d can be fired, and we can find $P^{M_1} = \{p_5\}$. Next, we compare the pre-sets of other activities in cl_3 with P^{M_k} to find deviation positions. Comparing ${}^\bullet f$ and P^{M_1}, we can find ${}^\bullet f = \{p_4\} \notin P^{M_1}$. Then f should be regarded as a logical input transition. We make $OP_l = {}^\bullet f = \{p_4\}$ and $NP_l = P^{M_1} = \{p_5\}$. At this time, we should add an arc from p_5 to f. According to [20], places p_4 and p_5 cannot contain tokens at the same time, so we have the logic input function $I'(f) = p_4 \otimes p_5$. Then transition f is fired, and we update $P^{M_2} = f^\bullet = \{p_6\}$. The pre-set of i is ${}^\bullet i = \{p_6\} \in P^{M_2}$, so i can be fired, and we update $P^{M_3} = \{p_8\}$. Then we compare ${}^\bullet j$ and P^{M_3}. Because ${}^\bullet j = \{p_7\} \notin P^{M_3}$, transition j should be modified as a logic input transition. OP_l denotes the pre-set of j, i.e., $OP_l = {}^\bullet j = \{p_7\}$. NP_l denotes P^{M_3}. An arc is added from p_8 to j, and we can get the logic input function of j: $I'(j) = p_7 \otimes p_8$. Then we can fire transition j according to P^{M_3} and I' (j). Finally, we get the repaired model as shown in Fig. 5.

Algorithm 2: Model Repair Algorithm Without Parallel Structures

```
Input: An event log $L$, and a process model $N_S = (PN, \alpha, M_i, M_f)$
Output: The repaired model denoted by a logic Petri net $LPN' = (P', T'; F', M', I', O')$
1. $LPN' \leftarrow N_S$;
2. Invoking Algorithm 1, and getting $CL$;
3. if ($\exists cl \in CL$ and $cl|(S_{RT} \downarrow T \cup S_{PT} \downarrow T) = \emptyset$), then
4.     $M[cl[1] > M'$, and we can get $P^{M_1}$;
5.     for ($k = 2; k++; k \leq |cl|$) do
6.         if (${}^\bullet cl[k] \notin P^{M_{k-1}}$), then
7.             $OP_l = {}^\bullet cl[k]$;
8.             $NP_l = P^{M_{k-1}}$;
9.             $F' \leftarrow F' \cup \{NP_l \rightarrow cl[k]\}$;
10.            $I'(cl[k]) = OP_l \otimes NP_l$;
11.            $M_k[cl[k] > M'_k$;
12.            $P^{M'_k} = cl[k]^\bullet$;
13.            $P^{M_k} = P^{M'_k}$;
14.        end if
15.        else
16.          $M_k[cl[k] > M'_k$;
17.          $P^{M'_k} = cl[k]^\bullet$;
18.          $P^{M_k} = P^{M'_k}$;
19.    end for
20. end if
21. return $LPN'$.
```

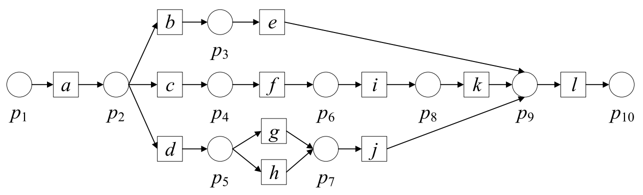

FIGURE 4. A process model N_{S2}.

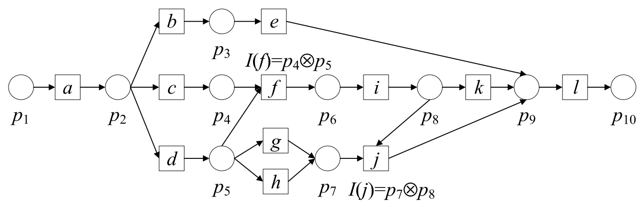

FIGURE 5. The repaired model of N_{S2} by our approach.

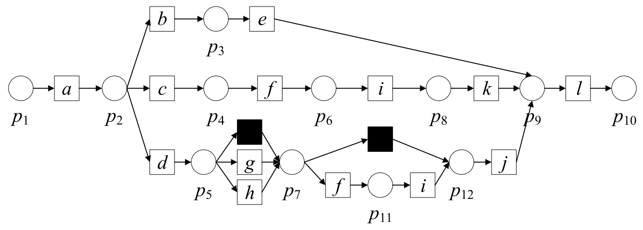

FIGURE 6. The repaired model of N_{S2} by Fahland's approach.

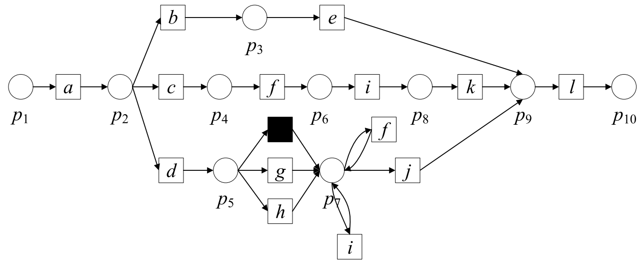

FIGURE 7. The repaired model of N_{S2} by Goldratt's approach.

From Fig. 6, we can see the model repaired by Fahland's method. This method first finds deviations according to the optimal alignment between σ_4 and Ns_2. Then it adds two invisible transitions, two repetitive transitions, and eight arcs to replay σ_4. Fig. 7 shows the model repaired by Goldratt's method. An invisible transition, two self-loops, and six arcs are added to the original model.

B. MODEL REPAIR WITH PARALLEL STRUCTURES

A choice deviation sub-trace cl contains activities in different choice branches. If a choice structure contains a parallel structure, then cl might contain parallel elements. Similar to Algorithm 2, we repair the original model once we find a deviation. Here we need to determine when a deviation appears.

Theorem 2: Let cl be a choice deviation sub-trace. There exists a deviation if $cl|(S_{RT} \downarrow T \cup S_{PT} \downarrow T) \neq \emptyset$ and $^\bullet cl[k+1] \notin (P^{M_{k-1}} \cup cl[k]^\bullet)$.

Proof: According to given definitions, $cl|(S_{RT} \downarrow T \cup S_{PT} \downarrow T) \neq \emptyset$ denotes that cl contains parallel elements. According to cl [1] and $cl[|cl|]$, we can determine a choice structure which cl belongs to, and the choice structure contains a parallel structure. For $\forall t \in T$, if $(cl[k], t) \cup (t, cl[k]) \notin S_{PT}$, then $P^{M_k} = cl[k]^\bullet$ after $cl[k]$ is fired. Otherwise, $P^{M_k} = (P^{M_{k-1}} - {}^\bullet cl[k]) \cup cl[k]^\bullet$. Therefore, a deviation exists if $^\bullet cl[k+1] \notin P^{M_k}$. Here, because of $P^{M_k} \in (P^{M_{k-1}} \cup cl[k]^\bullet)$, if $^\bullet cl[k+1] \notin P^{M_{k-1}} \cup cl[k]^\bullet)$, then a deviation exists between a trace and a process model. ■

For N_{S3} in Fig. 8, we assume that $c(g)$ is the first (last) activity of a parallel structure, and there may exist the following cases: (1) cl contains all activities in a parallel structure, e.g., $cl_4 = (b, c, e, f, g, h)$; (2) cl only contains the last activity in a parallel structure, e.g., $cl_5 = (b, g, h)$; (3) cl contains all activities on a parallel branch and the last activity, e.g., $cl_6 = (b, e, g, d)$; (4) cl contains activities on all parallel branches and the last activity, e.g., $cl_7 = (b, e, f, g, d)$. According to Theorem 2, the following algorithm repairs process models to replay these cases.

Algorithm 3 works as follows. Step 1 initializes LPN'' to N_S, and we can get a choice deviation sub-log by Step 2. By Step 3, we can obtain sub-traces containing parallel

Algorithm 3: Model Repair Algorithm With Parallel Structures

```
Input: An event log $L$, $S_{RT}$, $S_{PT}$, and a process model $N_S = (PN, \alpha, M_i, M_f)$
Output: The repaired model denoted by a logic Petri net $LPN'' = (P'', T''; F'', M'', I'', O'')$
1.  $LPN'' \leftarrow N_S$;
2.  Invoking Algorithm 1, and getting $CL$;
3.  if ($\exists cl \in CL$ and $cl|(S_{RT} \downarrow T \cup S_{PT} \downarrow T) \neq \emptyset$), then
4.      $P^{M_0} = {}^\bullet cl[1]$;
5.      for ($k = 1; k++; k \leq |cl|$) do
6.          if ($^\bullet cl[k+1] \notin (P^{M_{k-1}} \cup cl[k]^\bullet)$), then
7.              if ($(cl[k+1], t) \cup (t, cl[k+1]) \notin S_{PT}$ and $t \in T$), then
8.                  $M_k[cl[k] > M'_k, P^{M_k} = P^{M'_k}$;
9.                  $OP_l = {}^\bullet cl[k+1] - P^{M_k}$;
10.                 $NP_l = cl[k]^\bullet$;
11.                 $F'' \leftarrow F'' \cup \{NP_l \rightarrow cl[k+1]\}$;
12.                 $I''(cl[k+1]) = OP_l \otimes NP_l$;
13.             end if
14.             else if($(cl[k+1], t) \cup (t, cl[k+1]) \in S_{PT}$ and $t \in cl$), then
15.             $OP_l = cl[k]^\bullet$;
16.             $NP_l = {}^\bullet cl[k+1] \cup {}^\bullet t$;
17.             $F'' = F'' \cup \{cl[k] \rightarrow NP_l\}$;
18.             $O''(cl[k]) = OP_l \otimes NP_l$;
19.             $M_k\ [cl[k] > M'_k, P^{M_k} = P^{M'_k}$;
20.         end if
21.         else if($(cl[k+1], t) \cup (t, cl[k+1]) \in S_{PT}$ and $t \notin cl$), then
22.             $OP_l = cl[k]^\bullet$;
23.             $NP_l = {}^\bullet cl[k+1]$;
24.             $F'' \leftarrow F'' \cup \{cl[k] \rightarrow NP_l\}$;
25.             $O''(cl[k]) = OP_l \otimes NP_l$;
26.             $M_k[cl[k] > M'_k, P^{M_k} = P^{M'_k}$;
27.         end if
28.         end if
29.         else if($^\bullet cl[k+1] \in (P^{M_{k-1}} \cup cl[k]^\bullet)$ or $\nexists cl[k+1]$)
30.             $M_k[cl[k] > M'_k, P^{M_k} = P^{M'_k}$;
31.     end for
32. end if
33. return $LPN''$.
```

elements. Step 4 assigns P^{M_0} to the pre-set of cl [1]. Next, we fire transitions sequentially according to the ordering of events in the trace. We find deviation positions and repair the model by comparing the pre-set of transitions, the post-set of transitions and P^{M_k}. While traversing a sub-trace cl, if the pre-set of $cl\,[k+1]$ does not belong to $P^{M_{k-1}} \cup cl[k]^\bullet$, then a deviation appears, and $cl[k]$ or $cl[k+1]$ should be regarded as a logic transition (cf. Step 6). There are three cases for $cl[k]$ or $cl[k+1]$: (1) If $cl[k+1]$ does not have a parallel relation with any transitions, then $cl[k+1]$ is regarded as a logic input transition. According to the principle of token replaying, $cl[k]$ can be fired according to $P^{M_{k-1}}$, and we can get P^{M_k}. Then we use OP_l to denote ${}^\bullet cl[k+1] - P^{M_k}$, and NP_l to denote the post-set of $cl[k]$. Then we add arcs from NP_l to $cl[k+1]$, and the logic input function of $cl[k+1]$ is $I''(cl[k+1]) = OP_l \otimes NP_l$ (cf. Steps 7-13); (2) If $cl[k+1]$ have a parallel relation with transition $t \in cl$, which means that cl contains some activities which belong to different parallel branches. At this time, $cl[k]$ should be regarded as a logic output transition. OP_l denotes the post-set of $cl[k]$, and NP_l denotes the pre-sets of $cl[k+1]$ and t. Then we add arcs from $cl[k]$ to NP_l, and compute the logic output function of $cl[k]$: $O''(cl[k]) = OP_l \otimes NP_l$. According to $O''(cl[k])$ and $P^{M_{k-1}}$, we can fire $cl[k]$ and update P^{M_k} (cf. Steps 14-20); (3) If $cl[k+1]$ have a parallel relation with transition $t \notin cl$, which means that some activities in cl belong to a same parallel branch. At this time, $cl[k]$ is modified as a logic output transition. We make $OP_l = cl[k]^\bullet$ and $NP_l = {}^\bullet cl[k+1]$. Directed arcs are added from $cl[k]$ to NP_l, and the logic output function of $cl[k]$ is $O''(cl[k]) = OP_l \otimes NP_l$. Then $cl[k]$ can be fired, and P^{M_k} can be got (cf. Steps 21-27). If the pre-set of $cl[k+1]$ belongs to $P^{M_{k-1}} \cup cl[k]^\bullet$ or $\nexists cl[k+1]$, we can fire $cl[k]$ and update P^{M_k} according to the principle of token replaying (cf. Steps 29-30). Finally, we can get the repaired model denoted by logic Petri net LPN''.

Now, we use an example to illustrate the steps of Algorithm 3. Assuming that $L_3 = \{\sigma_5\} = \{<a, b, e, f, g, d, i>\}$, a process model N_{S3} ($M_{i1}(p_1) = 1$, i.e., $P^{M_{i3}} = \{p_1\}$) is shown in Fig. 8. According to Algorithm 1, $cl_8 = (b, e, f, g, d)$ is a choice deviation sub-trace. Choice activities b, e, f, g and d belong to different choice branches in N_{S3}. According to Algorithm 3, b is the first activity of cl_8, and we assume that $P^{M_0} = {}^\bullet b = \{p_2\}$. Next, we traverse cl_8 to find the deviation position. First, we compare the pre-set of e, the post-set of b and P^{M_0}. We can find that ${}^\bullet e = \{p_4\}$ is different from $P^{M_0} \cup b^\bullet = \{p_2, p_3\}$, i.e., ${}^\bullet e \notin (P^{M_0} \cup b^\bullet)$, and then b or e is regarded as a logic transition. Because of $(e, f) \in S_{PT}$ and $f \in cl$, we determine that b is a logic output transition. We make $OP_l = b^\bullet = \{p_3\}$ and $NP_l = {}^\bullet e \cup {}^\bullet f = \{p_4, p_5\}$. Then we add two arcs from b to p_4 and p_5, and compute the logic output function of b. According to [20], if p_3 contains a token, p_4 and p_5 do not contain tokens at a certain marking. Then we can obtain $O''(b) = p_3 \otimes (p_4 \wedge p_5)$. At this time, b is fired, and we update $P^{M_1} = \{p_4, p_5\}$ according to $O''(b)$ and P^{M_0}. Next, we compare ${}^\bullet f$ with $P^{M_1} \cup e^\bullet$, and we can find ${}^\bullet f = \{p_5\} \in (P^{M_1} \cup e^\bullet)$. Then e can be fired according to P^{M_1}, and we can get $P^{M_2} = \{p_5, p_6\}$. Comparing ${}^\bullet g$ with $P^{M_2} \cup f^\bullet$, we can get ${}^\bullet g = \{p_6, p_7\} \in (P^{M_2} \cup f^\bullet)$. Transition f is fired according to P^{M_2}, and $P^{M_3} = \{p_6, p_7\}$. Then we judge whether ${}^\bullet d$ belongs to $P^{M_3} \cup g^\bullet$. We can get ${}^\bullet d = \{p_3\} \notin (P^{M_3} \cup g^\bullet)$, and g or d is regarded as a logic transition. For $\forall t \in T$, we have $(d, t) \cup (t, d) \notin S_{PT}$, and then d is regarded as a logic input transition. At this time, transition g can be fired according to P^{M_3}, and we can get $P^{M_4} = \{p_8\}$. Then we make $OP_l = {}^\bullet d - P^{M_4} = \{p_3\}$, and $NP_l = g^\bullet = \{p_8\}$. We add an arc from p_8 to d. And the logic output function of d is $I''(d) = p_3 \otimes p_8$. d is the last activity of cl_3, and we can fire d according to P^{M_4} and $I''(d)$. Finally, we get the repaired model denoted by a logic Petri net as shown in Fig. 9.

The model repaired by Fahland's method is shown in Fig. 10. From the figure, we can find that four invisible transitions and a repetitive transition are added to the model. The model repaired by Goldratt's method is shown in Fig. 11. From this figure, we can find that three repetitive transitions as self-loops are added to the model. All models repaired by these three methods can replay the event log L_3.

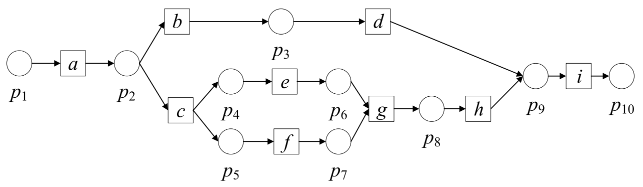

FIGURE 8. **A block-structured Petri net N_{S3}.**

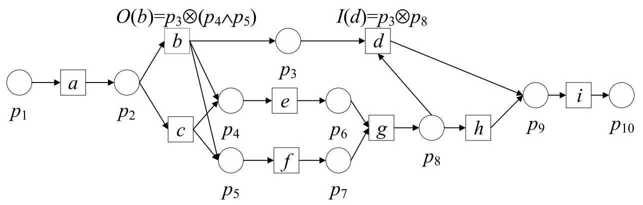

FIGURE 9. **The repaired model of N_{S3} by our approach.**

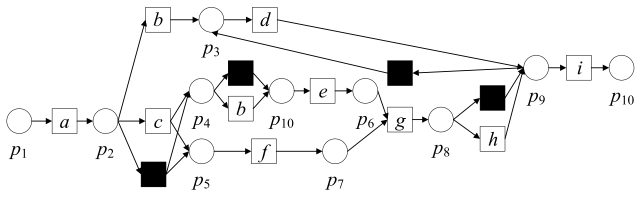

FIGURE 10. **The repaired model of N_{S3} by Fahland's approach.**

V. SIMULATION EXPERIMENTS

We carry out simulation experiments and comparative analysis in this section. The most recent model repair methods have good results for some particular structures (e.g., [11], [12]). The structures they repaired are different from this paper, and there is no comparability among them. Other recently model repair methods are based on different process models (e.g., BPMN) from Petri nets. Therefore, we select two most representative model repair methods for comparison.

TABLE 1. Twenty groups of detailed event log information.

Event logs	Traces	Events	Activities	Length	Event logs	Traces	Events	Activities	Length
L_1	113	1339	23	10-14	L_{11}	1219	14366	23	10-14
L_2	207	2475	23	10-14	L_{12}	1406	16861	23	10-14
L_3	312	3672	23	10-14	L_{13}	1631	19615	23	10-14
L_4	425	5107	23	10-14	L_{14}	1803	21506	23	10-14
L_5	518	6170	23	10-14	L_{15}	12043	24737	23	10-14
L_6	632	7582	23	10-14	L_{16}	2197	25213	23	10-14
L_7	757	9047	23	10-14	L_{17}	2405	28656	23	10-14
L_8	826	9852	23	10-14	L_{18}	2588	30828	23	10-14
L_9	932	11132	23	10-14	L_{19}	2859	34318	23	10-14
L_{10}	1025	12230	23	10-14	L_{20}	3002	36728	23	10-14

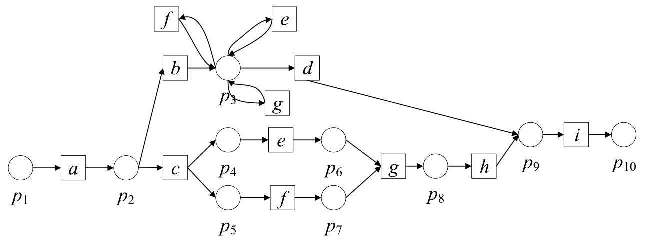

FIGURE 11. The repaired model of N_{S3} by Goldratt's approach.

The two methods used for comparison are Fahland's method [9] and Goldratt's method [10]. The former is implemented in the Process Mining Toolkit ProM6.6, available from http://www.promtools.org/prom6/, and the latter is implemented in the DOS window.

A. MODEL AND EXPERIMENT DATA

The model used in experiments is from a hospital in Tsingtao. Taking the diagnosis and treatment process of cancer patients as an example, we can get a model by mining original event logs, as shown in Fig. 12. Patients can reserve a doctor by the network in advance, and get a reservation number. On the other hand, they can register at the outpatient clinic when they go to the hospital. Then, the hospital should call their number by order. Next, patients could make an inquiry to a doctor. After that, the doctor decides which kind of examination to do according to the patients' condition. There are three types of examinations for patients to choose: (1) they can do ordinary CT and ESR; (2) they can do biochemical full set after enhanced CT is complete; or (3) they can do PET-CT, blood routine, and blood gas analysis in turn. After diagnosis, the patient needs to consult with the doctor to develop a corresponding treatment plan. If the patient's condition is mild, he (or she) can be treated in the outpatient clinic. They need to inject drugs, take medicines, and do laboratory testing. Patients with the severe disease need to do chemotherapy, and they are required routine observation according to the condition of chemotherapy. If the patient's condition is severe, he (or she) needs to have surgery. Some examinations are performed before the surgery. After a period of treatment and detection, the patient can leave the hospital.

All traces in event logs can be replayed at first. However, as the actual medical treatment process changes, there are some activities belonging to different choice branches contained in event logs, and the original model cannot replay these activities. For example, we can find that patients may choose ordinary CT and biochemical full set for examination, or blood routine and blood gas analysis may be performed after doing enhanced CT. Besides, patients may also need to take medicines, inject drugs and undergo laboratory testing during chemotherapy.

We obtain 20 event logs ($L_1 - L_{20}$) from a hospital system, and all the changes mentioned above are recorded in these logs. First of all, we manually removed the event logs which deviate from the original model seriously. The number of traces contained in logs ranges from 113 to 3002, and the event logs can be accessible at https://pan.baidu.com/s/1tQoA1NtzpLI3I8vppl1aNw. The number of traces, events, activities, and length range of traces are shown in Table 1.

B. MODEL REPAIRED EXPERIMENTS BASED ON LOGIC PETRI NETS

In this sub-section, we use Fahland's method, Goldratt's method, and our method to repair the original model as shown in Fig. 12. The event log L_{20} in Table 1 contains the most

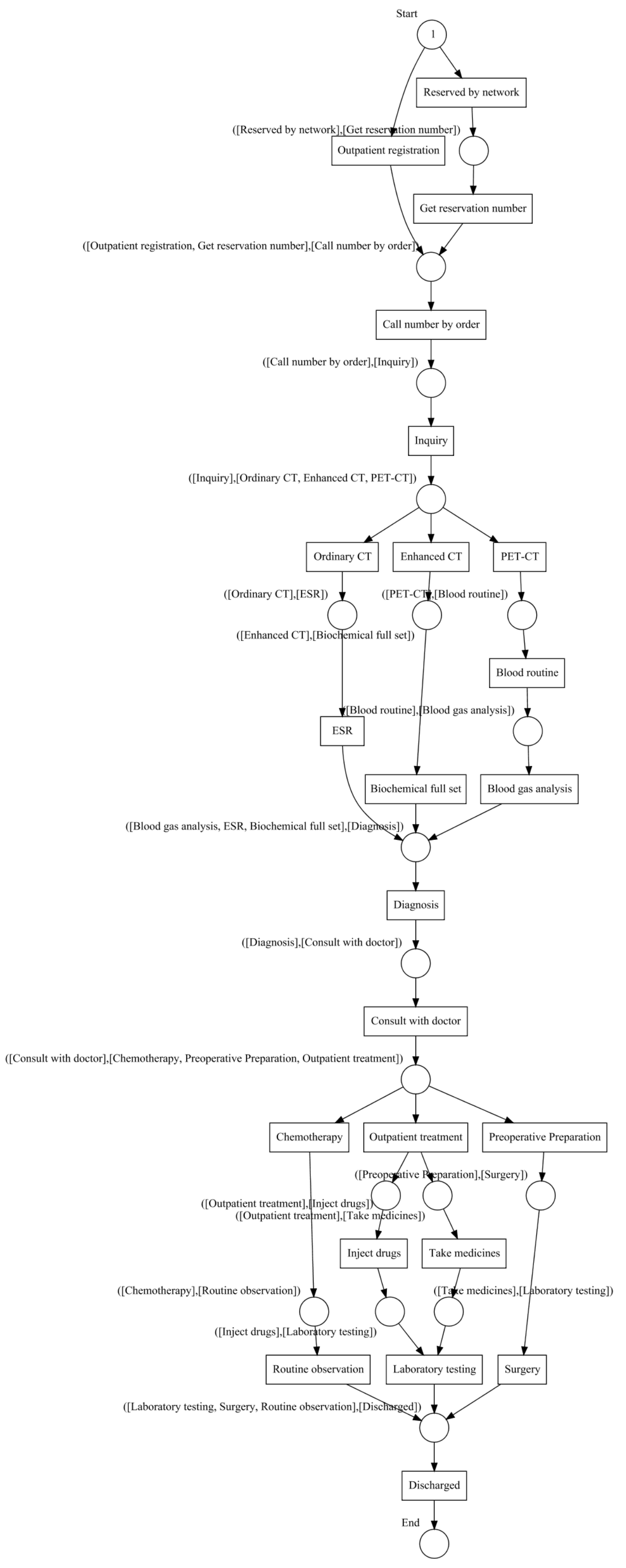
Start
1
Reserved by network
([Reserved by network],[Get reservation number])
Outpatient registration
Get reservation number
([Outpatient registration, Get reservation number],[Call number by order])
Call number by order
([Call number by order],[Inquiry])
Inquiry
([Inquiry],[Ordinary CT, Enhanced CT, PET-CT])
Ordinary CT
Enhanced CT
PET-CT
([Ordinary CT],[ESR])
([PET-CT],[Blood routine])
([Enhanced CT],[Biochemical full set])
Blood routine
([Blood routine],[Blood gas analysis])
ESR
Biochemical full set
Blood gas analysis
([Blood gas analysis, ESR, Biochemical full set],[Diagnosis])
Diagnosis
([Diagnosis],[Consult with doctor])
Consult with doctor
([Consult with doctor],[Chemotherapy, Preoperative Preparation, Outpatient treatment])
Chemotherapy
Outpatient treatment
Preoperative Preparation
([Preoperative Preparation],[Surgery])
([Outpatient treatment],[Inject drugs])
([Outpatient treatment],[Take medicines])
Inject drugs
Take medicines
([Chemotherapy],[Routine observation])
([Take medicines],[Laboratory testing])
([Inject drugs],[Laboratory testing])
Routine observation
Laboratory testing
Surgery
([Laboratory testing, Surgery, Routine observation],[Discharged])
Discharged
End

FIGURE 12. The process model for the treatment of cancer patients.

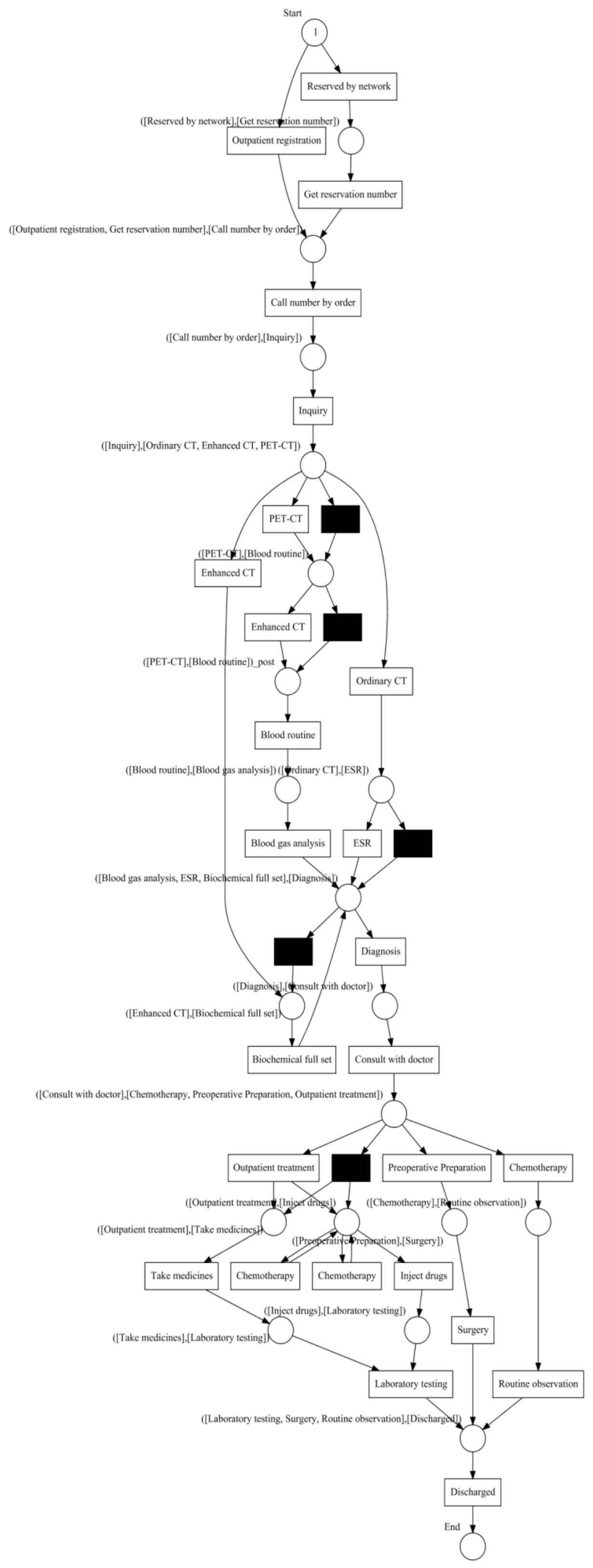
Start
1
Reserved by network
([Reserved by network],[Get reservation number])
Outpatient registration
Get reservation number
([Outpatient registration, Get reservation number],[Call number by order])
Call number by order
([Call number by order],[Inquiry])
Inquiry
([Inquiry],[Ordinary CT, Enhanced CT, PET-CT])
PET-CT
([PET-CT],[Blood routine])
Enhanced CT
Enhanced CT
([PET-CT],[Blood routine])_post
Ordinary CT
Blood routine
([Blood routine],[Blood gas analysis])
([Ordinary CT],[ESR])
Blood gas analysis
ESR
([Blood gas analysis, ESR, Biochemical full set],[Diagnosis])
Diagnosis
([Diagnosis],[Consult with doctor])
([Enhanced CT],[Biochemical full set])
Biochemical full set
Consult with doctor
([Consult with doctor],[Chemotherapy, Preoperative Preparation, Outpatient treatment])
Outpatient treatment
Preoperative Preparation
Chemotherapy
([Outpatient treatment],[Inject drugs])
([Chemotherapy],[Routine observation])
([Outpatient treatment],[Take medicines])
([Preoperative Preparation],[Surgery])
Take medicines
Chemotherapy
Chemotherapy
Inject drugs
([Inject drugs],[Laboratory testing])
([Take medicines],[Laboratory testing])
Surgery
Laboratory testing
Routine observation
([Laboratory testing, Surgery, Routine observation],[Discharged])
Discharged
End

FIGURE 13. The repaired model by Fahland's approach.

traces, and it may have the most comprehensive activities. Here, we use L_{20} as an experiment data for model repair. Figs. 13-15 show the repaired models by three methods, respectively.

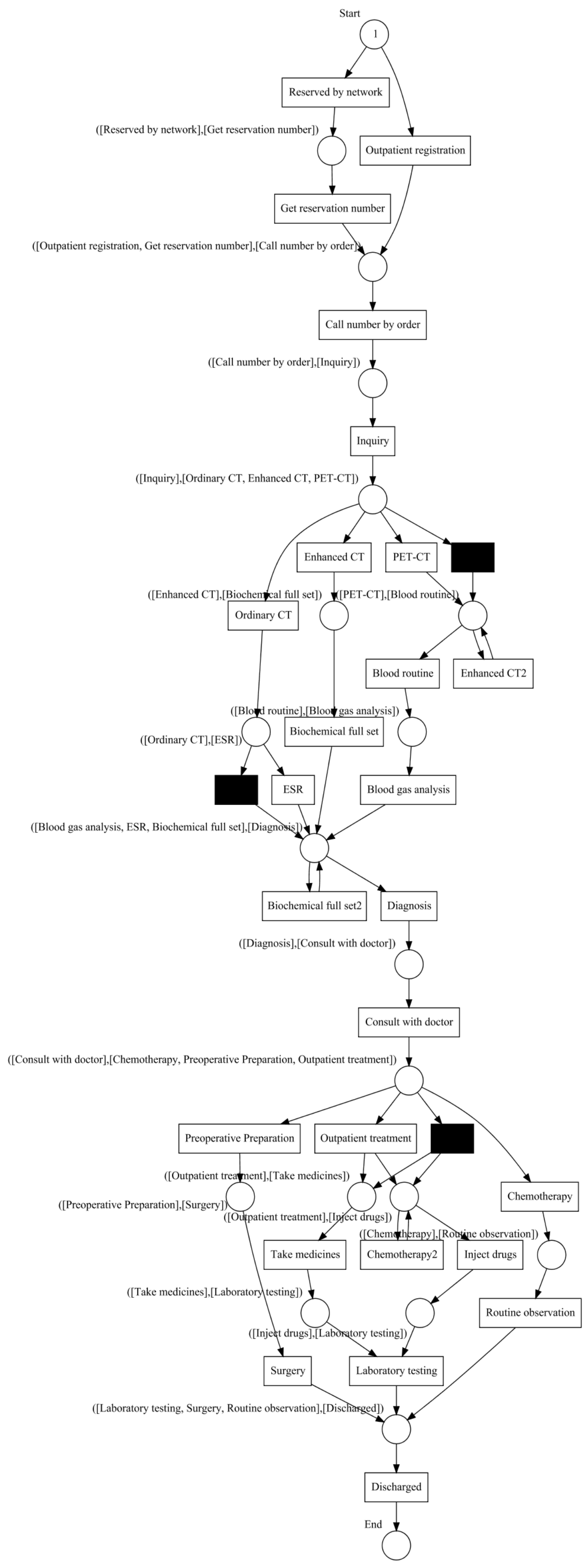
Start
Reserved by network
([Reserved by network],[Get reservation number])
Outpatient registration
Get reservation number
([Outpatient registration, Get reservation number],[Call number by order])
Call number by order
([Call number by order],[Inquiry])
Inquiry
([Inquiry],[Ordinary CT, Enhanced CT, PET-CT])
Enhanced CT
PET-CT
([Enhanced CT],[Biochemical full set])
([PET-CT],[Blood routine])
Ordinary CT
Blood routine
Enhanced CT2
([Blood routine],[Blood gas analysis])
([Ordinary CT],[ESR])
Biochemical full set
ESR
Blood gas analysis
([Blood gas analysis, ESR, Biochemical full set],[Diagnosis])
Biochemical full set2
Diagnosis
([Diagnosis],[Consult with doctor])
Consult with doctor
([Consult with doctor],[Chemotherapy, Preoperative Preparation, Outpatient treatment])
Preoperative Preparation
Outpatient treatment
([Outpatient treatment],[Take medicines])
Chemotherapy
([Preoperative Preparation],[Surgery])
([Outpatient treatment],[Inject drugs])
([Chemotherapy],[Routine observation])
Take medicines
Chemotherapy2
Inject drugs
([Take medicines],[Laboratory testing])
Routine observation
([Inject drugs],[Laboratory testing])
Surgery
Laboratory testing
([Laboratory testing, Surgery, Routine observation],[Discharged])
Discharged
End

FIGURE 14. The repaired model by Goldratt's approach.

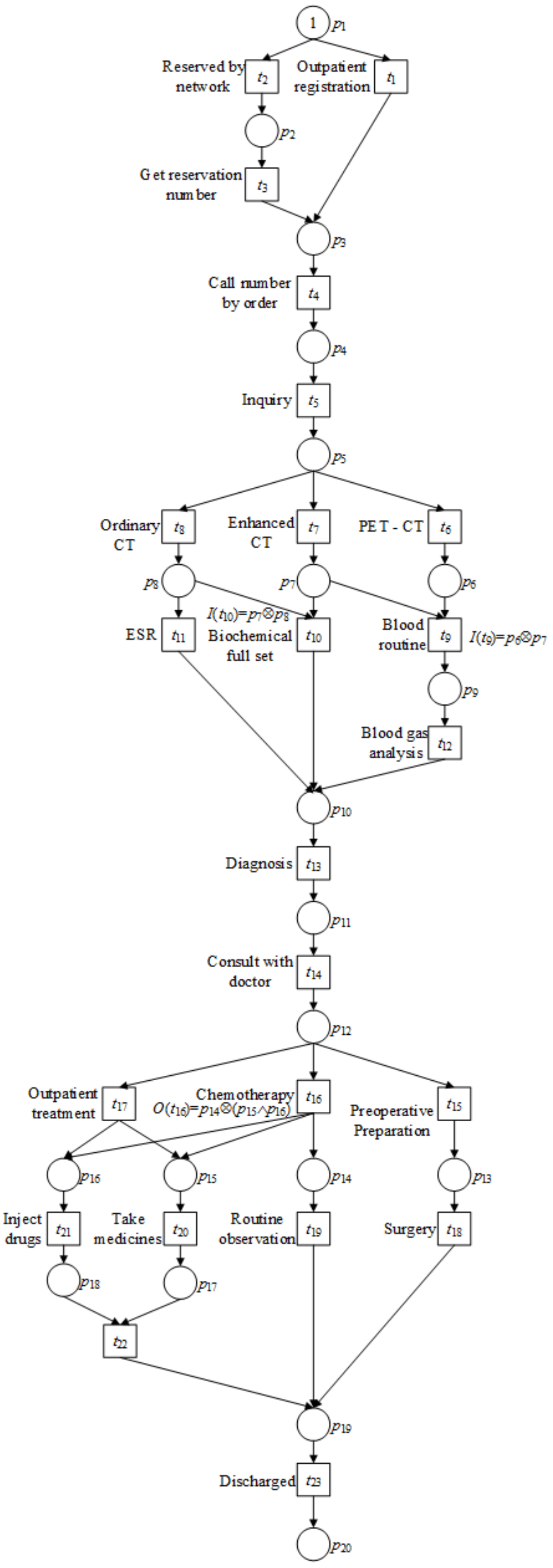

FIGURE 15. The repaired model by our approach.

Fig. 13 shows the model repaired by Fahland's method. This method obtains the deviations based on the optimal alignments between the given event logs and the original model. We can collect sub-logs according to the moves on log in optimal alignments. Then we mine the corresponding sub-process and add it in the form of a self-loop to the original model to complete the model repair. For activities belonging to different choice branches in event logs, this method can find deviations according to alignments. Then it adds invisible transitions and sub-processes to repair the original model. Comparing the repaired model with the original one, the changes are as follows: five invisible transitions are added to the model; a repetitive transition is added to a choice branch; two self-loops are added to a place. The model repaired by Fahland's method can replay all event logs.

The model repaired by Goldratt's method is shown in Fig. 14. This method adds a single transition as a self-loop or invisible transition to the model according to some constraints. For event logs that contain activities on different choice branches, this method inserts invisible transitions and self-loops in the original model to replay these event logs. From Fig, 14, we can find that compared with the original model, the repaired model obtained by Goldratt's method adds three invisible transitions and three repetitive transitions as self-loops.

When we repair the original model by our method, we can get a choice pre-post pair set $CPS = \{(p_5, p_{10}), (p_{12}, p_{19})\}$, a choice branch tuple set $CTS = \{(t_6, t_9, t_{12}), (t_7, t_{10}), (t_8, t_{11})\}, (t_{15}, t_{18}), (t_{16}, t_{19}), (t_{17}, t_{20}, t_{21}, t_{22})\}$, and a parallel transition pair set $S_{PT} = \{(t_{20}, t_{21})\}$. Then we can get a choice deviation sub-log $CL = \{cl_9, cl_{10}, cl_{11}, cl_{12}\} = \{<t_8, t_{10}>, <t_7, t_9, t_{12}>, <t_{16}, t_{20}, t_{21}, t_{22}>, <t_{16}, t_{21}, t_{20}, t_{22}>\}$ by Algorithm 1. We can find that there are two choice structures in the original model. The choice structure with (p_5, p_{10}) as the choice pre-post pair does not contain a parallel structure. Another choice structure contains a parallel structure.

Now, we use Algorithms 2 and 3 to repair the two choice structures respectively according to the choice deviation sub-log CL. We use choice deviation sub-traces cl_9 and cl_{10} to repair the choice structure without a parallel structure. According to Algorithm 2, the first activity t_8 in cl_9 is fired at first, and we can get $P^{M_1} = \{p_8\}$. Then we find deviation position by comparing the pre-set of $cl_9[2]$ and P^{M_1}. The pre-set of $cl_9[2]$ is ${}^\bullet cl_9[2] = {}^\bullet t_{10} = \{p_7\}$, which is different from P^{M_1}. At this time, $cl_9[2] = t_{10}$ is regarded as a logic input transition. We use OP_l to denote the pre-set of t_{10}, i.e., $OP_l = {}^\bullet t_{10} = \{p_7\}$, and $NP_l = P^{M_1} = \{p_8\}$. We add an arc from p_8 to t_{10}. Places p_7 and p_8 cannot contain a token at the same time, and the logic input function of t_{10} is $I'(t_{10}) = p_7 \otimes p_8$. Then t_{10} can be fired according to P^{M_1} and $I'(t_{10})$. When we repair the original model with cl_{10}, it is similar to cl_9. We can find that t_9 is also a logic input transition, and $OP_l = {}^\bullet t_9 = \{p_6\}$, $NP_l = \{p_7\}$. Only one of p_6 and p_7 contains a token at the same time, and then we can get the logical input function of t_9: $I'(t_9) = p_6 \otimes p_7$. And the rest of activities in cl_{10} can be fired.

Choice deviation sub-traces cl_{11} and cl_{12} are used to repair the choice structure containing a parallel structure. By Algorithm 3, we have $P^{M_0} = {}^\bullet cl_{11}[1] = {}^\bullet t_{16} = \{p_{12}\}$. Next, we traverse cl_{11} to find the deviation position. The pre-set of $cl_{11}[2]$ is ${}^\bullet cl_{11}[2] = {}^\bullet t_{20} = \{p_{15}\}$, and the post-set of $cl_{11}[1]^\bullet = t_{16}^\bullet = \{p_{14}\}$. By comparing ${}^\bullet cl_{11}[2]$, $cl_{11}[1]^\bullet$ and P^{M_0}, we can find that ${}^\bullet t_{20} \notin (P^{M_0} \cup t_{16}^\bullet)$, and then t_{16} or t_{20} needs to be regarded as a logic transition. Because $(t_{20}, t_{21}) \in S_{PT}$ and $t_{21} \in cl_{11}$, we can determine that t_{16} is a logic output transition. Then we use OP_l to denote the post-set of t_{16}, i.e., $OP_l = t_{16}^\bullet = \{p_{14}\}$, and $NP_l = {}^\bullet t_{20} \cup {}^\bullet t_{21} = \{p_{15}, p_{16}\}$. Two arcs are added from t_{16} to NP_l. According to [20], only p_{14} (or p_{15} and p_{16}) can contain a token at some marking, and we can get $O''(t_{16}) = p_{14} \otimes (p_{15} \wedge p_{16})$. At this time, $cl_{11}[1] = t_{16}$ can be fired, and we can update $P^{M_1} = \{p_{15}, p_{16}\}$ according to P^{M_0} and $O''(t_{16})$. Then we compare ${}^\bullet cl_3[3]$ with $P^{M_1} \cup cl_{11}[2]^\bullet$. Because ${}^\bullet cl_{11}[3] = {}^\bullet t_{21} = \{p_{16}\} \in (P^{M_1} \cup cl_{11}[2]^\bullet)$, t_{20} can be fired and we can get $P^{M_2} = \{p_{16}, p_{17}\}$. Similarly, we can fire t_{21} according to P^{M_2} and get $P^{M_3} = \{p_{17}, p_{18}\}$. t_{22} is the last activity of cl_3, so we fire t_{22} according to P^{M_3}, and $P^{M_4} = \{p_{19}\}$. When we repair the original model with cl_{12}, it is similar to cl_{11}. Finally, we get the repaired model denoted by a logic Petri net as shown in Fig. 15. Comparing the original model, there are four arcs added to the repaired model. No repetitive activities exist in the repaired model, and the model repaired by our method preserves the structure of the original model.

The method proposed in this paper focuses on adding bridges among choice branches. We assume that all deviations occur in choice structures and there are no new activities or repetitive activities in event logs. The repaired model denoted by a logic Petri net does not have the corresponding process tree, and it cannot be repeatedly repaired directly. However, the actual process may change at any time. If the repaired model cannot replay new event logs, we need to repair the model again. At this time, we can find choice structures and new choice deviation sub-logs based on the process tree of the original model. Then we conduct repeated repair on the repaired model according to the new choice deviation sub-logs.

C. MODEL EVALUATION

Taking 20 event logs $L_1 - L_{20}$ in Table 1 as an example, we compare our method with two methods, i.e., Fahland's method and Goldratt's method. The three approaches are compared from the aspects of fitness, precision, and simplicity in this sub-section. Note that, logic Petri nets are the extension of Petri nets, and they have good applications in studying the theoretical properties of nets because of their logic expressions [16]–[18]. The equivalence between logic Petri nets and the safe inhibition Petri nets is proved in [19]. Therefore, we compute the fitness and precision between

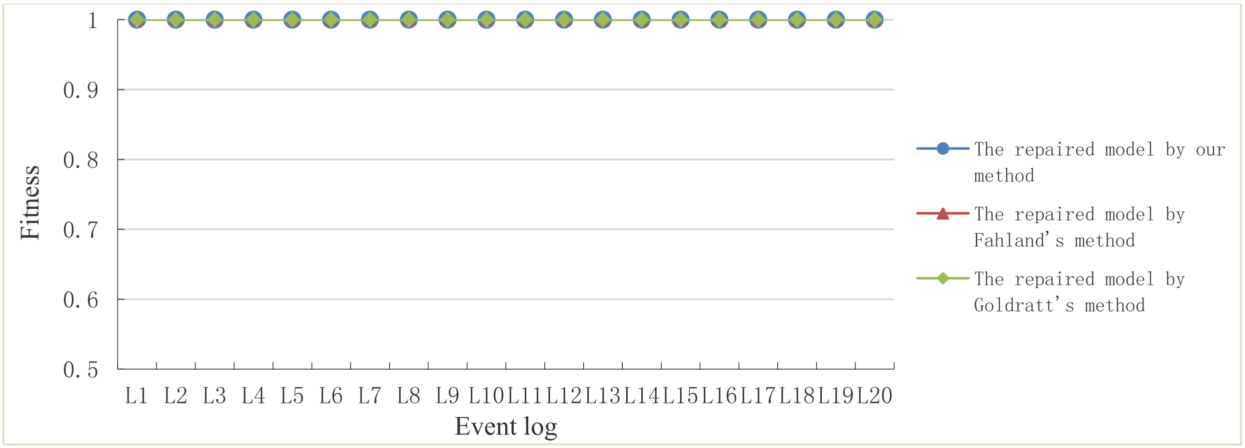

FIGURE 16. The comparison results of fitness between our method and other methods.

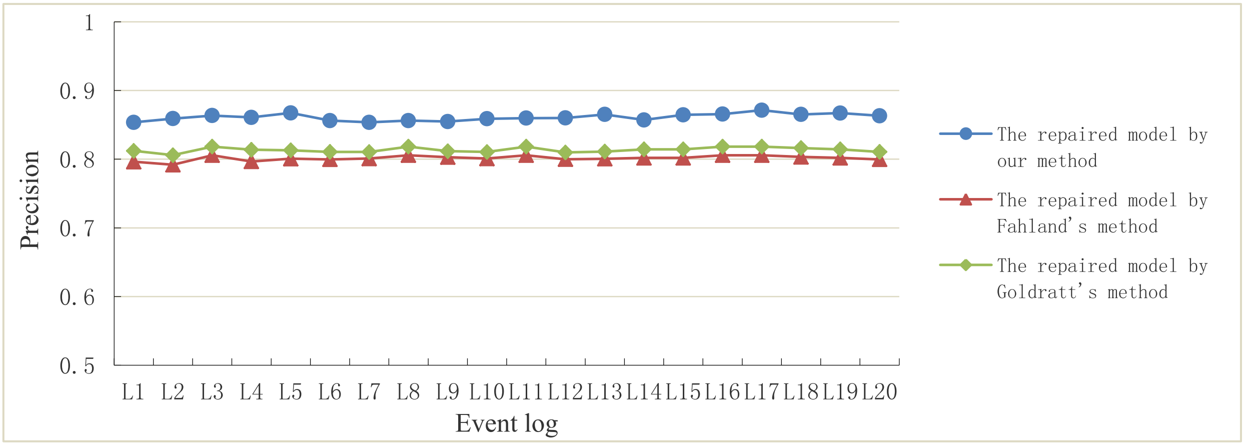

FIGURE 17. The comparison results of precision between our method and other methods.

TABLE 2. The comparison of simplicity for the original model in Fig. 12.

Model	Number of added places $\lvert P\rvert$	Number of added transitions $\lvert T\rvert$	Number of added invisible transitions $\lvert\tau\rvert$	Number of added arcs $\lvert F\rvert$
The repaired model by our approach	0	0	0	4
The repaired model by Fahland's approach	1	3	5	17
The repaired by Goldratt's approach	0	3	3	13

event logs and the repaired model according to the formulas provided in [5].

Fig. 16 shows the comparison results of fitness among our method, Fahland's method, and Goldratt's method. Fitness is the most important metric to evaluate the quality of a process model. If a model can replay activities in event logs, the model has a better fitness. If a model can reproduce all traces in event logs, then the fitness of the model is 1. From Fig. 16, we can find that the fitness of Fahland's method, Goldratt's method and our method are all 1 under different numbers of event logs. That is, the model repaired by these three methods can replay all traces in event logs.

The comparison results of precision among the three methods are shown in Fig. 17. Precision means that the activities which cannot be observed in event logs are not allowed to happen in a process model. From Fig. 17, we can find that the precision value does not change much with the increasing number of traces in event logs. Our method has a higher precision value than that of other two methods in different amounts of event logs. Repetitive transitions or self-loops may reduce the precision of the repaired model.

Simplicity means that the repaired model should be as simple as possible. To compare three methods' simplicity, the main criteria we focus on are the number of added places, transitions, invisible transitions, and arcs. Compared with the original model in Fig. 12, Table 2 shows the number of added places, transitions, invisible transitions, and arcs in repaired models by three methods, respectively. For two choice structures of the original model, Fahland's method adds some invisible transitions and sub-processes to guarantee choice branches can be replayed. Thus, the number of added places, transitions, invisible transitions, and arcs are respectively 1, 3, 5, and 17. Goldratt's method adds repetitive transitions as self-loops and invisible transitions to the original model. Then, the number of added transitions, invisible transitions and arcs are respectively 3, 3 and 13. Our method can repair choice structures by adding four arcs.

For models with choice structures, Petri net-based repair methods do not add bridges among choice branches, and the repaired model has some invisible transitions and repetitive transitions. Logic Petri net-based methods can solve such problems by adding bridges among choice branches. They not only simplify net structures, but also improve the precision of the model by using logic expressions instead of repetitive transitions. What's more, logic Petri net-based models have been applied to modeling and analysis of workflow, e-commerce, medical treatment and other fields.

VI. CONCLUSIONS

In this paper, we propose a model repair method based on logic Petri nets. By adding bridges among choice branches, the model repaired by our method can replay activities in different branches. Combining with process tree, we can find the pre-post pair of a choice structure. Thus, we can find choice deviation sub-logs. By comparing pre-sets (post-sets) of transitions with a set of places containing one token at marking M_k, we can find the deviation positions, and repair the model in two different ways. The repaired model can correctly describe the actual process. The structure of the repaired model is similar to the original one. Through simulation experiments, we compare our method with other benchmark methods. The precision of our method is higher than other methods. Also, the repaired model denoted by logic Petri net has good simplicity. The method in this paper focuses on repairing a choice structure to make the model replay activities in different branches. In the future, we will propose some methods to repair other complex structures based on logic Petri nets, and propose new model repair methods based on other models [14], [15] which has logic expressions and can be directly repaired again.

REFERENCES

[1] W. M. P. van der Aalst, *Process Mining: Data Science in Action*. Berlin, Germany: Springer, 2016, pp. 2–49.

[2] W. M. P. van der Aalst and B. F. van Dongen, "Discovering Petri nets from event logs," in *Transactions on Petri Nets and Other Models of Concurrency VII*, vol. 7. Berlin, Germany: Springer, 2013, pp. 372–422.

[3] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *WIREs Data Mining Knowl. Discovery*, vol. 2, no. 2, pp. 182–192, 2012.

[4] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin, Germany: Springer, 2011, pp. 23–146.

[5] A. Adriansyah, "Aligning observed and modeled behavior," Ph.D. dissertation, Dept. Math. Comput. Sci., Eindhoven Univ. Technol., Eindhoven, The Netherlands, 2014.

[6] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.

[7] L. Wen, W. M. P. van der Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 145–180, 2007.

[8] L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang, and J. Sun, "Mining process models with prime invisible tasks," *Data Knowl. Eng.*, vol. 69, no. 10, pp. 999–1021, Oct. 2010.

[9] D. Fahland and W. M. P. van der Aalst, "Model repair—Aligning process models to reality," *Inf. Syst.*, vol. 47, no. 1, pp. 220–243, 2015.

[10] A. Polyvyanyy, W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. T. Wynn, "Impact-driven process model repair," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 4, p. 28, 2016.

[11] X. Zhang, Y. Du, L. Qi, and H. Sun, "Repairing process models containing choice structures via logic Petri nets," *IEEE Access*, vol. 6, pp. 53796–53810, 2018.

[12] H. Qi, Y. Du, L. Qi, and L. Wang, "An approach to repair Petri net-based process models with choice structures," *Enterprise Inf. Syst.*, vol. 12, nos. 8–9, pp. 1149–1179, 2018.

[13] Y. H. Xu, Y. Y. Du, W. J. Luan, L. Qi, and H. C. Sun, "Repairing process models with logical concurrent and casual relations via logical Petri nets," *IEEE Access*, vol. 6, pp. 56340–56355, 2018.

[14] W. M. P. van der Aalst and A. H. M. ter Hofstede, "YAWL: Yet another workflow language," *Inf. Syst.*, vol. 30, no. 4, pp. 245–275, 2004.

[15] W. M. P. van der Aalst, "The application of Petri nets to workflow management," *J. Circuits, Syst. Comput.*, vol. 8, no. 1, pp. 21–66, 1998.

[16] Y. Du, L. Qi, and M. Zhou, "Analysis and application of logical Petri nets to e-commerce systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 44, no. 4, pp. 468–481, Apr. 2014.

[17] Y. Du, L. Qi, and M. Zhou, "A vector matching method for analysing logic Petri nets," *Enterprise Inf. Syst.*, vol. 5, no. 4, pp. 468–481, 2011.

[18] W. Luan, L. Qi, and Y. Du, "Composition of logical Petri nets and compatibility analysis," *IEEE Access*, vol. 5, pp. 9152–9162, 2017.

[19] Y. Y. Du and B. Q. Guo, "Logic Petri nets and equivalency," *Inf. Technol. J.*, vol. 8, no. 1, pp. 95–100, 2009.

[20] Y. Y. Du *et al.*, "An approach of process mining based on logic Petri nets," *Acta Electron. Sinica*, vol. 44, no. 11, pp. 2742–2751, 2016.

[21] W. Liu *et al.*, "Soundness analytics of composed logical workflow nets," *Int. J. Parallel Program.*, pp. 1–16, Oct. 2017, doi: 10.1007/s10766-017-0536-8.

[22] L. Qi, M. Zhou, and W. Luan, "Emergency traffic-light control system design for intersections subject to accidents," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 1, pp. 170–183, Jan. 2016.

[23] L. Qi, M. Zhou, and W. Luan, "A two-level traffic light control strategy for preventing incident-based urban traffic congestion," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 13–24, Jan. 2018.

[24] C. Liu, "Automatic discovery of behavioral models from software execution data," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 4, pp. 1897–1908, Oct. 2018.

[25] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Scalable process discovery with guarantees," in *Enterprise, Business-Process and Information Systems Modeling* (Lecture Notes in Business Information Processing), vol. 214. Cham, Switzerland: Springer, 2015, pp. 85–101.

[26] N. Q. Wu and M. Zhou, "Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 2, pp. 446–454, Apr. 2012.

[27] N. Wu and M. Zhou, "Schedulability analysis and optimal scheduling of dual-arm cluster tools with residency time constraint and activity time variation," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 1, pp. 203–209, Jan. 2012.

[28] N. Wu, F. Chu, C. Chu, and M. Zhou, "Petri net modeling and cycle-time analysis of dual-arm cluster tools with wafer revisiting," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 1, pp. 196–207, Jan. 2013.

[29] N. Wu, M. Zhou, and Z. Li, "Short-term scheduling of crude-oil operations: Enhancement of crude-oil operations scheduling using a Petri net-based control-theoretic approach," *IEEE Robot. Autom. Mag.*, vol. 22, no. 2, pp. 64–76, Jun. 2015.

[30] N. Wu, M. Zhou, L. Bai, and Z. Li, "Short-term scheduling of crude oil operations in refinery with high-fusion-point oil and two transportation pipelines," *Enterprise Inf. Syst.*, vol. 10, no. 6, pp. 581–610, May 2016.

[31] L. Bai, N. Wu, Z. Li, and M. Zhou, "Optimal one-wafer cyclic scheduling and buffer space configuration for single-arm multicluster tools with linear topology," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 10, pp. 1456–1467, Oct. 2016.
[32] F. Yang, N. Wu, Y. Qiao, M. Zhou, and Z. Li, "Scheduling of single-arm cluster tools for an atomic layer deposition process with residency time constraints," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 3, pp. 502–516, Mar. 2017.
[33] Q. Zhu, M. Zhou, Y. Qiao, and N. Wu, "Petri net modeling and scheduling of a close-down process for time-constrained single-arm cluster tools," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 3, pp. 389–400, Mar. 2018.
[34] Y. Qiao, N. Wu, F. Yang, M. Zhou, and Q. Zhu, "Wafer sojourn time fluctuation analysis of time-constrained dual-arm cluster tools with wafer revisiting and activity time variation," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 4, pp. 622–636, Apr. 2018.
[35] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, "A genetic algorithm for discovering process trees," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2012, pp. 1–8.
[36] Z. Ma, Z. Li, and A. Giua, "Characterization of admissible marking sets in Petri nets with conflicts and synchronizations," *IEEE Trans. Autom. Control*, vol. 62, no. 3, pp. 1329–1341, Mar. 2017.
[37] Z. Jiang, Z. Li, N. Wu, and M. Zhou, "A Petri net approach to fault diagnosis and restoration for power transmission systems to avoid the output interruption of substations," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2566–2576, Sep. 2017.
[38] F. Yang, N. Q. Wu, Y. Qiao, and R. Su, "Polynomial approach to optimal one-wafer cyclic scheduling of treelike hybrid multi-cluster tools via Petri nets," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 1, pp. 270–280, Jan. 2018.
[39] N. Q. Wu, M. Zhou, and L. P. Bai, "Control-theoretic and model-based scheduling of crude oil transportation for refinery industry," in *Proc. IEEE Int. Conf. Robot. Automat.*, Hong Kong, May/Jun. 2014, pp. 3273–3278.
[40] S. Zhang, N. Wu, Z. Li, T. Qu, and C. Li, "Petri net-based approach to short-term scheduling of crude oil operations with less tank requirement," *Inf. Sci.*, vol. 417, pp. 247–261, Nov. 2017.
[41] S. Wang, D. You, and C. Seatzu, "A novel approach for constraint transformation in Petri nets with uncontrollable transitions," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 48, no. 8, pp. 1403–1410, Aug. 2018.
[42] D. You, S. Wang, and M. Zhou, "Computation of strict minimal siphons in a class of Petri nets based on problem decomposition," *Inf. Sci.*, vols. 409–410, pp. 87–100, Oct. 2017.
[43] S. Wang, M. Gan, M. Zhou, and D. You, "A reduced reachability tree for a class of unbounded Petri nets," *IEEE J. Autom. Sinica*, vol. 2, no. 4, pp. 345–352, Oct. 2015.
[44] D. Xiang, G. Liu, C. Yan, and C. Jiang, "Detecting data-flow errors based on Petri nets with data operations," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 1, pp. 251–260, Jan. 2018.
[45] G. Liu and C. Jiang, "Petri net based model checking for the collaborativeness of multiple processes systems," in *Proc. IEEE 13th Int. Conf. Netw., Sens., Control (ICNSC)*, Würzburg, Germany, Apr. 2016, pp. 1–6.
[46] H. Qi, Y. Du, and W. Liu, "Process model repairing method based on reachable markings," *J. Shandong Univ. Sci. Technol., Nature Sci.*, vol. 36, no. 1, pp. 118–124, Feb. 2017.
[47] G. J. Liu and C. J. Jiang, "Net-structure-based conditions to decide compatibility and weak compatibility for a class of inter-organizational workflow nets," *Sci. China Inf. Sci.*, vol. 58, no. 7, pp. 1–16, 2015.
[48] Y. Teng, Y. Du, L. Qi, and W. Luan, "A logic Petri net-based method for repairing process models with concurrent blocks," *IEEE Access*, vol. 7, pp. 8266–8282, 2018, doi: 10.1109/ACCESS.2018.2890070.
[49] C. Gu, Z. W. Li, N. Q. Wu, M. Khalgui, T. Qu, and A. Al-Ahmari, "Improved multi-step look-ahead control policies for automated manufacturing systems," *IEEE Access*, vol. 6, no. 1, pp. 68824–68838, 2018.

YUHUA XU received the B.S. degree from the Shandong University of Science and Technology, Qingdao, China, in 2017, where she is currently pursuing the M.S. degree with the College of Computer Science and Engineering. Her current research interests include process mining, Petri nets, and workflow.

YUYUE DU received the B.S. degree from Shandong University, Jinan, China, in 1982, the M.S. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 1991, and the Ph.D. degree in computer application from Tongji University, Shanghai, China, in 2003. He is currently a Professor with the College of Information Science and Engineering, Shandong University of Science and Technology, Qingdao, China. He has taken in over ten 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 140 papers in domestic and international academic publications, and they are embodied over 80 times by SCI and EI and cited over 270 times by others. His research interests include formal engineering, Petri nets, real-time systems, Web services, and workflows. He is a member of the Professional Committee of Petri Nets of the China Computer Federation.

LIANG QI (S'16–M'18) received the B.S. degree in information and computing science and the M.S. degree in computer software and theory from the Shandong University of Science and Technology, Qingdao, China, in 2009 and 2012, respectively, and the Ph.D. degree in computer software and theory from Tongji University, Shanghai, China, in 2017. From 2015 to 2017, he was a Visiting Student with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA. He is currently a Lecturer of computer science and technology with the Shandong University of Science and Technology. He has authored over 25 technical papers in journals and conference proceedings, including the IEEE Transactions on System, Man and Cybernetics: Systems, the IEEE Transactions on Intelligent Transportation Systems, and the IEEE/CAA Journal of Automatica Sinica. His current research interests include Petri nets, discrete event systems, process mining, and optimization algorithms. He received the Best Student Paper Award-Finalist in the 15th IEEE International Conference on Networking, Sensing and Control, in 2018.

WENJING LUAN (S'16) received the B.S. and M.S. degrees from the Shandong University of Science and Technology, Qingdao, China, in 2009 and 2012, respectively, and the Ph.D. degree in computer software and theory from Tongji University, Shanghai, China, in 2018. In 2017, she was a Visiting Student with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA. She is currently a Lecturer of computer science and technology with the Shandong University of Science and Technology. Her current research interests include location-based social networks, data mining, recommender systems, and intelligent transportation systems. She received the Best Student Paper Award-Finalist in the 13th IEEE International Conference on Networking, Sensing and Control, in 2016.

LU WANG received the B.S. and Ph.D. degrees from the Shandong University of Science and Technology, Qingdao, China, in 2013 and 2018, respectively. She is currently a Lecturer of computer science and technology with the Shandong University of Science and Technology. Her current research interests include process mining, workflow, and Petri nets.