IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 9, 2018, accepted May 29, 2018, date of publication June 1, 2018, date of current version June 20, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2843137

An Approach for Repairing Process Models Based on Logic Petri Nets

XIZE ZHANG[1], YUYUE DU[1], LIANG QI[1], AND HAICHUN SUN[2]
[1]College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China
[2]College of Information Technology and Network Security, People's Public Security University of China, Beijing 100038, China

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

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFC0803700, in part by the National Natural Science Foundation of China under Grant 61170078 and Grant 61472228, in part by the Taishan Scholar Construction Project of Shandong Province, in part by the Natural Science Foundation of Shandong province under Grant ZR2018MF001, in part by the Scientific Research Foundation of the Shandong University of Science and Technology for Recruited Talents under Grant 2017RCJJ044, and in part by the Key Research and Development Program of Shandong Province under Grant 2018GGX101011.

ABSTRACT Process mining technology can extract process knowledge from event logs that are generated from information systems. It can construct a process model by mining the event logs. The model usually needs to be repaired to accurately describe the business process realized by the information system. The existing methods for repairing process models cannot enhance efficiently some model consistency metrics, such as the fitness, precision, and simplicity. Thus, a new model repair approach is proposed based on an extended Petri net named logic Petri net in this paper. It can improve the model's fitness and precision comparing with the existing work. First, it builds process models via logic Petri nets. Next, approaches are proposed to repair the process models containing a causal relation and a concurrent relation, respectively. Specifically, a precursor set and a successor set of activities are defined and the relation of the elements in each of them is determined. Finally, we give some cases related to a thoracic surgery process in a hospital and conduct experiments to illustrate the correctness and effectiveness of the proposed approach.

INDEX TERMS Process mining, consistency checking, logic Petri net, model repair.

I. INTRODUCTION

In the current dynamic business world, business success is increasingly dependent on the ability to respond to environmental changes in a quick, flexible and cost-effective way [3]. Information systems are thus increasingly used to realize operational business processes. Meanwhile, numerous recorded events called event logs are generated in the systems, which help companies to extract valuable information during the business process. Process mining techniques have been used to extract non-trivial and useful information from event logs [1], [2]. There are mainly three types of process mining techniques: process discovery, consistency checking and enhancement. The process discovery can use an event log to generate a process model. There are some techniques that can automatically construct a process model (e.g., a Petri net or a BPMN model) from an event log [1], [4]–[9]. The consistency checking can be used to verify if a constructed process model conforms to the real business processes. The aim of consistency checking is to compare a known process model with the event logs generated by the model, and check if a sequence of events (called a trace) can be repeated in the model. Then we can analyze the consistency of model behaviors and log behaviors. Some techniques based on consistency checking can detect when and where a process is changed [10]. The consistency checking can be explained from two perspectives: the model does not reflect the actual behavior (''model error''), and the reality deviates from the expected model (''event log error''). The process enhancement is to extend or improve an existing process model with the generated event logs.

Some metrics have been proposed to determine the quality of a process model, such as fitness, simplicity, precision and generalization [1]. Fitness is an important quality metric which is used to ensure the occurrence of behaviors in event logs. In this work, the fitness of a process model is called ideal, if the model can completely repeat the behaviors of event logs. Simplicity requires that a process model with a simple structure can execute the behaviors in the event logs. Precision means that a process model does not allow activities to happen if they are not observed in an event log.

Generalization claims that a process model is able to reproduce the activities that will happen in the future.

When the event logs and the corresponding process model are inconsistent during the consistency checking, i.e., the values of quality metrics are low, we need to repair the process model. The difference between a process model and an actual operational process in event logs is identified by using a suitable deviation recognition approach. Models can be thus repaired by adding or deleting elements or operations at appropriate locations.

The existing model repair approaches ensure a good simplicity but fail to get a high fitness value. Besides, the approaches cannot enhance efficiently other consistency quality metrics, such as generalization and precision [4]. And when the process models need to be repaired, the existing approaches usually collect some sub-logs and sub-processes. We adopt the divide-and-conquer idea in this work. We define a causal relation and a concurrent relation for the process models. When such relations exist and some deviations occur in the alignments between the process model and the event log, the repaired model by the existing approaches may have a lower precision value. A lot of useless traces generated by the repaired model cannot be observed in the event log. Thus, we propose efficient approaches to repair process models with such relations, respectively. This work has the following contributions:

(1) In order to confirm where the deviations appear in a process model, we propose a new approach by identifying new activities.

(2) The proposed approaches can repair process models with the causality and concurrent relations, respectively. Specifically, we define a precursor set and a successor set of the activities, and the relations among the activities in each set. We can construct the logic relation functions. By this way, the process model can be repaired.

(3) The correctness and effectiveness of the proposed approach are illustrated by simulations.

The organization of this paper is as follows. Related work is discussed in Section II. Some basic concepts are reviewed in Section III. Sections IV and V present a new approach to repair process models with a causal relation and a concurrent relation, respectively. Experimental results are given in Section VI. Section VII concludes the work.

II. RELATED WORK

In the studies of process mining, the existing enterprises and organizations generally obtain the corresponding process model according to their business processes [11]. However, it is hard for business processes [4] to adapt to the development of enterprises and organizations with the increasing development of the business. For an event log, we can use a process mining algorithm to obtain a process model. At present, scholars have proposed many process mining algorithms, such as the α algorithm [4], which mines process models based on the order of activities. However, a correct model must not include repetitive activities, invisible transitions and specific structures. Therefore, a number of variants of the α algorithm have emerged. The $\alpha\#$ algorithm proposed in [12] makes some improvements to delete the invisible transitions. Wen *et al.* [13] put forward a corresponding process mining approach based on α algorithm. The aim is to solve the problem that the structure of non-free choice cannot be judged. However, because of some external factors, it is often difficult for enterprises to have business processes that are compatible with the development of enterprises.

The existing consistency checking approaches proposed in [14] mainly include alignment and footprint comparisons. Its aim is to compare a known process model with the event log generated from the information system and check whether each trace can be repeated in the model. And the aim of consistency checking [15], [16] is to find deviations between activities allowed by a model and those observed in an event log, but does not focus on fixing deviations. Moreover, the existing position deviation and model repair approaches do not repair the model well. Fahland and van der Aalst [17] obtains the deviation through an alignment approach, and then collects these deviations of sub-logs and digs out sub-processes by using the existing process mining algorithm. Those sub-logs can be added to the original model in a self-loop structure. The approach only takes into account the fitness metric. Not only added sub-processes can occur in an infinite loop, but also there will be different parts describing the same behavior in the repaired model. The approach greatly reduces the precision and increases the complexity of the model, which will make the repair results unsatisfied. Besides, the repaired model cannot correctly represent the structure of the model and the logic relations between transitions. Thus, the model may have no similarity with the original one, especially that if the original is created manually [17]. Besides, there is also a workflow model repair approach based on some mirroring matrices [18]. There are also many approaches of repairing models to describe real processes. i.e., mining process variants [3], dealing with concept drifts in process mining [10] and aligning process models to reality [17]. But these approaches have a bad repair effect.

With the increasingly widespread application of Petri nets in complex system modeling, there are many extensions of Petri nets. A typical one is called a logic Petri net, it can easily describe and analyze batch data processing and state uncertainty in real-time collaborative work systems [19]. In [20], a mining approach based on logic transition is proposed for the process mining in complex systems. It can well describe the relation between parallel activities. An effective approach based on the logic Petri net is proposed to repair the model in this work.

III. PRELIMINARIES

Petri nets (PNs) are a well-founded model applied widely to the description and analysis of a complex discrete event dynamic system because of their graphical function and mathematical rigor. It is particularly convenient to describe the system's order, conflict, concurrency, and synchroniza-

tion. This section briefly introduces the basic definitions of tuple, trace, event log, Petri net [1], [21]–[24], [29]–[32], workflow net [25], [33], and the properties of the Petri net-based model, such as alignment [26], [27], soundness [28], and other related concepts.

Definition 1 (Multiple Set): Let S be a set. A multi-set D over S is a function D: $S \to \mathbb{N}_+$, where $\mathbb{N}_+$ represents a set of positive integers. B(S) represents the set of all multi-sets over S.

Definition 2 (Tuples): Let S be a set. $m = (a_1, a_2, \ldots, a_n) \in S \times \ldots \times S$ is a tuple consisting of n elements, and $\pi_i(m)$ is the i-th element of m.

Definition 3 (Traces and Event Logs): Let $\mathcal{A}$ be the set of all activities. $A \subseteq \mathcal{A}$ represents a set; A^* represents a finite sequence set over the set $\mathcal{A}$; $\sigma \in \mathcal{A}^*$ is called a trace; $L \in \mathcal{B}(A^*)$ is a multiple set of traces called an event log, and $\&(\sigma)$ represents the set of all activities in trace σ.

For example, $A = \{a, b, c, d, e\}$ is a set, and $\sigma =< b, c, a, e, d >$ is a trace with $\&(\sigma) = \{b, c, a, e, d\}$.

Definition 4 (Pre-Set and Post-Set): $N = (P, T; F)$ is called a net, where P is a finite set of places, T is a finite set of transitions, and F is a set of directed arcs from places to transitions or from transactions to places. For $x \in P \cup T$,

$$^\bullet x = \{y | y \in P \cup T \wedge (y, x) \in F\}$$
$$x^\bullet = \{y | y \in P \cup T \wedge (x, y) \in F\}$$

where $^\bullet x$ denotes the pre-set of x, $x^\bullet$ denotes the post-set of x, and $^\bullet x \cup x^\bullet$ denotes the extension of x.

Definition 5 (Petri Net): A four-tuple $\Sigma = (P, T; F, M)$ is called *Petri* net, where

1) $N = (P, T; F)$ is a net;
2) M: $P \to N_+$ is called a marking of N; and
3) Transition firing rules:
   1) For transition $t \in T$, if $\forall p \in^\bullet t$: $M(p) \geq 1$, t is enabled denoted by $M\ [t >$; and
   2) If $M\ [t >$, it means that the transition t can occur under the marking M, and after the transition t is fired, a new marking M' is generated, denoted by $M\ [t > M'$, where

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

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

Definition 6 (Workflow Net): $WFN = (P, T; F, M, i, o)$ is called a workflow net, where

(1) $\sum = (P, T; F, M)$ is a Petri net;
(2) There is a start place $i \in P$, and $^\bullet i = \emptyset$;
(3) There is an end place $o \in P$, and $o^\bullet = \emptyset$; and
(4) For $\forall x \in P \cup T$, x lies on a path from i to o.

Tokens can be used to represent the pre-conditions and the post-conditions of tasks in Petri nets. Here, we use tokens to represent resources in workflow nets.

Definition 7 (Alignment): An alignment of the tuple $(a, t) \in \gamma$ is defined as follows:

(1) If $a \in A$ and $t =\gg$, then it is recorded as the log activity;
(2) If $a =\gg$ and $t \in T$, then it is recorded as the model activity;
(3) If $a \in A$ and $t \in T$, then it is recorded as the synchronous activity; and
(4) Otherwise, it is recorded as the illegal activity.

Definition 8 (Move Sequence): Let $A \subseteq \mathcal{A}$ be a set of activities. $\sigma \in A^*$ is a trace on A, $PN = (P, T; F, M)$ is a Petri net. The tuple $(a, t) \in A \gg \times T \gg \{(\gg, \gg)\}$ is called a move, where $\gg$ represents there is no move. An alignment $\gamma \in (A \gg \times T \gg)^*$ is a move sequence between a trace denoted by σ and the model PN, where

(1) $\pi_1(\gamma)_{\downarrow A} = \sigma$ denotes the trace sequence generated by a move sequence;
(2) $m_i \xrightarrow{\pi_2(\gamma)_{\downarrow T}} m_f$, where the move sequence of the model generates a complete firing sequence denoted by $\pi_2(\gamma)_{\downarrow T}$.

$\Gamma_{\sigma,PN}$ denotes the set of all alignments between the trace σ and the model PN.

Definition 9 (Soundness): Let $WFN = (P, T; F, M, i, o)$ be a Workflow net with i as the start place and o as the end place. WFN is sound if the following conditions hold:

(1) For any reachable marking M from i, there is a transition sequence from i to an end marking, where the end marking is a marking that o has at least one token;
(2) A start marking can lead to an end marking after firing a sequence of transitions; and
(3) There is no dead transition which cannot be fired.

Definition 10 (Process Tree): Let $\mathcal{A}$ be the set of all activities. $A \subseteq \mathcal{A}$ represents a set, and $\oplus$represents a given operator set, and τ is an implicit transition, where

(1) $a \in A \cup \{\tau\}$ is a process tree; and
(2) If all of $PT_1, \ldots, PT_n (n > 0)$ are process trees, then $\oplus(PT_1, \ldots, PT_n)$ is also a process tree.

There are four types of operators: $\times$ represents the selection relation, and only one subtree corresponding to the operator will occur; $\rightarrow$ represents the sequential relation, and the sub-tree corresponding to the operator happens in sequence; represents the loop relation, and if PT_1 represents the loop body, then $PT_2, \ldots, PT_n$(n≥2) represents the loop path; $\wedge$ represents the parallel relation, which represents that each sub-tree corresponding to the operator will occur.

Definition 11 (Logic Petri Net): A five-tuple $LN = (P, T; F, I, O)$ is called a logic net, and a six-tuple $LPN = (LN, M)$ is called a logic Petri net, where

(1) P is a finite place set;
(2) $T = T_I \cup T_O \cup T_D$ is a finite transition set, and $T \cap P = \varnothing$, $T \cup P \neq \varnothing$. If $t \in T_I \cap T_O$, $^\bullet t \cap t^\bullet = \varnothing$,

where

a) T_I represents a logic input transition set, for $\forall t \in T_I$, the input place $^\bullet t$ of t will be affected by the logic input function $f_I(t)$;

b) T_O represents a logic output transition set, for $\forall t \in T_o$, the output place $t^\bullet$ of t will be affected by the logic function $f_O(t)$;

c) T_D represents a transition set in classic Petri nets;

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

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

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

(6) $M : P \rightarrow \{0,1\}$ is the marking function of Petri nets, for $\forall p \in P, M(p)$ represents the token numbers in p; and

(7) Transition firing rules:

a) For $\forall t \in T_I$, $I(t) = f_I(t)$. If $f_I(t)|_M = .T.$, namely when the marking is M, the logic input function $f_I(t)$ of the logic input transition t is true, then logic input transition t can be fired, it can be recorded as $M[t > M'$, and $\in \forall p^\bullet t$, $M'(p) = 0$; for $\forall p \in t^\bullet$, $M'(p) = 1$; for $\forall p \notin t^\bullet \cup {}^\bullet t$, $M'(p) = M(p)$;

b) For $\forall t \in T_o$, $O(t) = f_O(t)$. If $\forall p \in {}^\bullet t$, $M(p) = 1$, then logic output transition t can be fired, and for $\in \forall p^\bullet t$, $M'(p) = 0$. For $\forall p \notin t^\bullet \cup {}^\bullet t$, $M'(p) = M(p)$; For $\forall p \in t^\bullet$, it needs to meet $f_O(t)|_{M'} = .T.$, namely when the marking is M', the logic output function $f_o(t)$ of the logic output transition t is true; and

c) For $\forall t \in T_D$, transition firing rules are consistent with classical Petri nets.

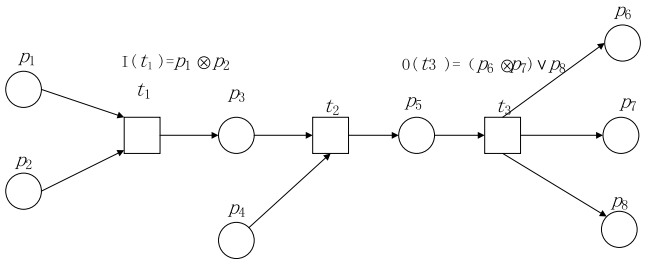

FIGURE 1. A logic Petri net model LN_1.

Figure 1 gives an example of a simple logic Petri net denoted by LN_1, where t_2 is a classic transition, t_1 is a logic input transition, and $I(t_1) = p_1 \otimes p_2$ is a logic input function of t_1. When t_1 is fired, p_1 and p_2 do not contain tokens at the same time. t_3 is a logic output transition, and $O(t_3) = p_6 \vee p_7 \vee p_8$ is its logic output function. When t_3 is fired, there will be five kinds of cases: (1) both p_6 and p_8 contain a token; (2) both p_7 and p_8 contain a token; (3) p_6 contains a token; (4) p_7 contains a token; and (5) p_8 contains a token.

IV. THE LOGIC REPAIR OF THE MODELS WITH THE CAUSALITY RELATION

In this section, we give the relevant definition of the logic Petri net-based repairing approach by considering the model with a causal relation. When we do an alignment between activities in a log and transitions in the model, there will be many deviations, and the repaired models obtained by different repair approaches are different. The basic steps of the repair approach are given in this section: new activities in the log are derived from a projection, which is derived from the alignment between activities in the log and transitions in the model. First of all, we introduce a concept of the projection from activities in the log to transitions in the model.

Algorithm 1 The Discovery Algorithm of a New Activity Set Act_{new}

Input: Complete event log $L \in \mathcal{B}(A^*)$, Workflow net $WFN = (P, T; F, M, i, o)$

Output: The set of new activities Act_{new}

1. $Actnew \leftarrow \varnothing$, $T_M \leftarrow \varnothing$;
2. FOR ($i = 1$; $i <= length$; i++) DO
3. $T_M \leftarrow T_M \cup \{t_i\}$;
4. FOR EACH $\sigma \in L$ DO
5. FOR ($a_j \in \sigma, j = 1; j <= length; j$++) DO
6. $\eta(a_j, t_k)$;
7. IF ($t_k \notin T_M$)
8. $Act_{new} \leftarrow Act_{new} \cup \{a_j\}$;
9. RETURN Act_{new};

Definition 12 (Activity Projection): For a log $L \in B(A^*)$, and $\forall \sigma \in L$, if $a \in \&(\sigma)$, then the projection from a in the log trace to the corresponding transition t in the model is denoted by $\eta(a, t)$.

The concept is used to get new activities in the logs, which are obtained from the projection from activities in the log to transitions in the model. We default that the marks of activities in the log are the same as transitions in the model. Next, we give the discovery algorithm of a new activity set.

Example 1: Figure 2 shows a Petri net model N_1 that has a simple sequential relation. There is an event log $L_1 = \{< t_1, t_6, t_3, t_4, t_5 >, < t_1, t_6, t_4, t_5 >, < t_1, t_2, t_6, t_3, t_4, t_5 >, < t_1, t_2, t_6, t_4, t_5 >, < t_1, t_2, t_3, t_4, t_5 >\}$.

Definition 13 (Precursor and Successor): Let $L \in \mathcal{B}(A^*)$ be a log. For $\forall \sigma \in L$, if an activity $a \in \&(\sigma)$ and the position index of a in the trace is i, the activity a_l at the position with index $i - 1$ is called the precursor of a; and the activity a_r at the position with index $i + 1$ is the successor of a.

The set of new activities $Act_{new} = \{t_6\}$ can be collected by projecting activities in the trace with transitions in the model in Figure 2. Once new activities are obtained, we can get their precursor and successor activities in each trace. We give some definitions about the relations between two activities next.

For example, in a trace $< t_1, t_6, t_3, t_4, t_5 >$, t_1 is the precursor of t_6 and t_3 is the successor of t_6.

Definition 14 (Ordering Relations): Let $L \in B(A^*)$ be an event log on the set A, and $\sigma \in L$ be a trace in the event log with $a, b \in \&(\sigma)$. We have the following relations between two activities a and b:

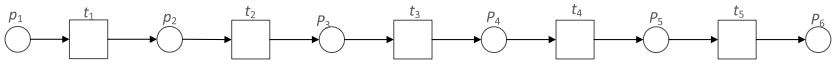

FIGURE 2. A Petri net model denoted by N_1 with a simple sequential relation.

(1) Follow relation $>_L$: $a >_L b$ if there is a trace $\sigma =< t_1, t_2, t_3, \ldots, t_n >, i \in \{1, \ldots, n-1\}$: $\sigma \in L, t_i = a$ and $t_{i+1} = b$;
(2) Causal relation $\rightarrow_L$: $a \rightarrow_L b$ if any trace $\sigma \in L$, $a, b \in \&(\sigma)$: $a >_L b$ and $b \not>_L a$;
(3) Selective relation $\#_L$: $a\#_L b$ if any trace $\sigma \in L$: $a \in \&(\sigma)$ and $b \notin \&(\sigma)$ or $b \in \&(\sigma)$ and $a \notin \&(\sigma)$;
(4) Concurrent relation $||_L$: $a||_L b$ if there are two traces σ, $\sigma' \in L$ for $a, b \in \&(\sigma)$: $a > b$ and for $a, b \in \&(\sigma')$: $b > a$.

Notation $>_L$ denotes all pairs of the activities with a follow relations in L. $\rightarrow_L$ contains all pairs of the activities with a causal relations in L, e.g., activity b is followed directly by activity a, but a is never followed directly by b in L. $\Rightarrow_L$ contains all pairs of the activities with a causal relations in L. $\#_L$ contains all the pairs of the activities with a selective relations in L, e.g., the relation between a and b is a selective relation, and if a is executed and b is not executed in a same process instance. $||_L$ contains all pairs of the activities with concurrent relations, e.g., the relation between a and b is a concurrent relation, and if a is executed, b will be executed in a same process instance. For $\forall a, b \in A$, there is a determined relation between a and b, e.g., $a \rightarrow b$, $a \Rightarrow b$, $a\#b$ or $a||b$. Notation $|A|$ denotes the number of elements in a set A. Any two activities must satisfy one of the four relations. The following is a relations determination algorithm for event logs.

Algorithm 2 A Ordering Relations Determination Algorithm for Event Logs

Input: Complete event log $L \in \mathrm{B}(A^*)$
Output: The set of log-based ordering relations denoted by R'
1. $R \leftarrow \emptyset, R' \leftarrow \emptyset$
2. FOR EACH $\sigma_m \in L$ DO
3. $R \leftarrow R \cup \{a_i >_L a_{i+1}\}$;
4. IF ($a \in \sigma_m$ and $b \notin \sigma_m$ OR $b \in \sigma_m$ and $a \notin \sigma_m$) THEN
5. $R' \leftarrow R' \cup \{a\#_L b\}$;
6. IF ($a >_L b$ and $b \not>_L a$)THEN
7. $R' \leftarrow R' \cup \{a \rightarrow_L b\}$;
8. IF ($a >_L b$ and $b >_L a$)THEN
9. $R' \leftarrow R' \cup \{a||_L b\}$;
10. Return R';

Algorithm 2 only obtains the relations between activities in the event logs. Combining new activities in the log with the predecessor set, a new relations determination algorithm based on the precursor and successor sets of new activities is given below based on Algorithm 2.

Algorithm 3 A Relations Determination Algorithm Based on Precursor and Successor Sets

Input: Complete event log $L \in \mathrm{B}(A^*)$
Output: The relations set R_{pre} based on the precursor set S_{pre} of activities and the relations set R_{subs} based on the successor set S_{subs} of activities
1. $R_{pre} \leftarrow \emptyset, R_{subs} \leftarrow \emptyset, S_{pre} \leftarrow \emptyset, S_{subs} \leftarrow \emptyset$;
2. By Algorithm 1, get the set of new activities Act_{new};
3. FOR EACH $\sigma_m \in L$ DO
4. FOR ($a_{loc} \in \sigma$, loc=1; $j <= length$; loc++) DO
5. $R \leftarrow R \cup \{a_i >_L a_{i+1}\}$;
6. IF ($a_{loc} \in Act_{new}$) DO
7. $S_{pre} \leftarrow S_{pre} \cup \{a_{loc-1}\}, S_{subs} \leftarrow S_{subs} \cup \{a_{loc+1}\}$;
8. FOR EACH $\sigma_m \in L, a, b \in S_{pre}$ DO
9. By Algorithm 2, RETURN R_{pre};
10. FOR EACH $\sigma_m \in L, a, b \in S_{subs}$ DO
11. By Algorithm 2, RETURN R_{subs};

For example, for the event log L_1 and the Petri net model N_1 in Example 1, we can obtain the precursor set $S_{pre} = \{t_1, t_2\}$, the successor set $S_{subs} = \{t_3, t_4\}$, the precursor relations set $R_{pre} = \{t_1 \rightarrow t_2\}$, and the successor relations set $R_{subs} = \{t_3 \rightarrow t_4\}$ by using Algorithms 1-3. We can use the relation to find those logic transitions in the Petri net and these logic input (output) functions [20]. The repaired model is transformed into a logic Petri net model. We give the definition of the logic preorder matrix next.

Definition 15 (Logic Preorder Matrix): Let $PN = (P, T; F, M)$ be a *Petri* net is obtained by a log L. For $t \in T$, $T_{bef}|t$ is a set of logic preorder transitions. The logic preorder matrix of t is defined as $(\boldsymbol{AI}|t)[n][n](n = |T_{bef}|t|)$. For $i, j \in [0, n]$, if $i \neq j$, then $(\boldsymbol{AI}|t_1)[i][j] = lr(i, j)$; Else, $(\boldsymbol{AI}|t_1)[i][j] = \varnothing$.

In [20], $PN = (P, T; F, M)$ is a Petri net, which are obtained by the log L. We can define the logic relation $lr(a, b)$ between these activities. There are four kinds of relations between two activities a and b:

(1) $a \otimes b$: When the logic output transition t is fired, for a and b, it only allows one of them to be fired in the set of preorder transitions;
(2) a and b: When the logic output transition t is fired, both a and b will be fired in the set of preorder transitions, but they will not be fired simultaneously.
(3) $a \wedge b$. When the logic output transition t is fired, the corresponding successor places of a and b will be fired simultaneously.
(4) $a \vee b$. When the logic output transition t is fired, a or b is fired in the set of preorder transitions. The specific definitions are not described in detail.

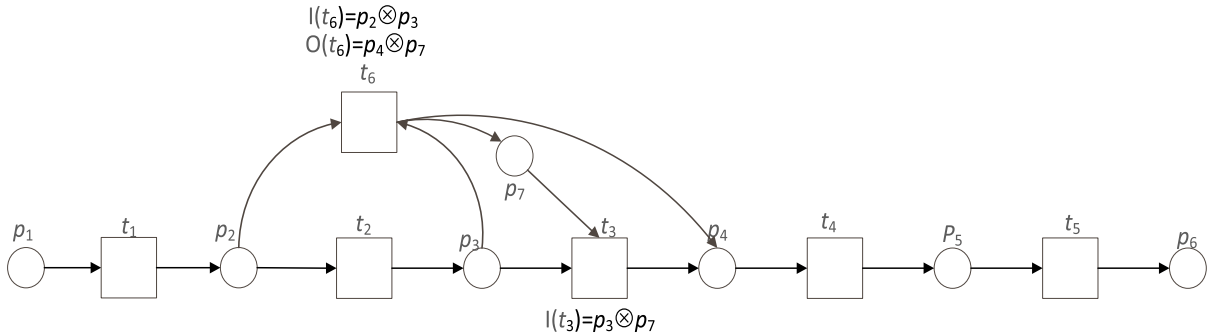

FIGURE 3. The repaired model by our approach.

The logic relation of the transition is transformed into a logic preorder matrix $(\boldsymbol{AI}|t)[n][n](n = |T_{bef}|t|)$, and then the function of preorder transitions $\lambda(T', \boldsymbol{AI}|t)$ is obtained, and then the logic input function $I(t) = pl(\lambda)$ is obtained. The precursor and successor sets of new activities in the trace are used in the paper, and they actually are similar to the set of preorder transitions. Due to the logic structure of logic Petri nets based on causal relations, the logic structure is relatively simple. Only one of logic functions can be obtained through the logic relation of transitions in the precursor set. Thus, it can be converted into a logic input function without requiring a logic precursor matrix. When the logic transition relation is $a \rightarrow b$, its corresponding places are similar with $a \otimes b$, and thus it can be transformed into $a \otimes b$. The following is a logic repair algorithm based on the causal relation.

According to [20], the logic relation between transitions based on logic Petri nets is given. We can obtain the new activity t_6 in the log by using Algorithm 1. Then we get the precursor set $S_{pre} = \{t_1, t_2\}$, the successor set $S_{subs} = \{t_3, t_4\}$, the precursor relations set $R_{pre} = \{t_1 \rightarrow t_2\}$, and the successor relations set $R_{subs} = \{t_3 \rightarrow t_4\}$. Therefore, the logic relation between these two transitions in the precursor set is $lr(t_1, t_2) = t_1 \rightarrow t_2$, and the logic input function is $I(t_6) = p_2 \otimes p_3$. If these pre-set places of the new activity's successor set are same as the post-set places of the new activity's precursor set, then a new place needs to be added. The aim is to express the logic output function. Hence a place P_7 needs to be added, and similarly, we have $I(t_3) = p_3 \otimes p_7$ and $O(t_6) = p_4 \otimes p_7$. If new activities are inserted into the model, additional invisible transitions need to be added, which will increase the complexity of the model. The repaired model is shown in Figure 3.

The repaired model based on the Fahland's approach is shown in Figure 4. The idea of this approach is to use the alignment to discover the deviation between transitions in the model and activities in the log. We can collect sub-logs, and then mine sub-logs by the existing mining algorithm. Finally, the sub-process is added as a self-loop to appropriate places in the original model. Because of the addition of self-loops in the model, the sub-process can be repeated indefinitely in the repaired model. In the actual process, such sub-process may not be allowed to repeat, which will generate a lot of unnecessary traces, and greatly reduce the accuracy of the model. However, the model obtained by applying the repair approach in this paper is more simple and accurate, the structure is clearer, and the precision of the model is very high. It does not appear that different parts can describe the same behavior, and no self-loop occurs at the same time. Table 1 gives a comparison of added elements based on the two repair approaches in Example 1.

This paper aligns activities in the log with transitions in the model, and then collects precursor activities. Thus the precursor set can be constructed. And then the relation of precursor-set activities in the model can be determined. Therefore, in the model, when these transitions corresponding to precursor-set

Algorithm 4 Model Repair of Logic Petri Nets Based on the Causal Relation

Input: Complete event log $L \in B(A^*)$, Workflow net $WFN = (P, T; F, M, i, o)$

Output: Repaired model of the logic Petri net $LPN' = (P', T'; F', I', O', M')$

1. $N \leftarrow LPN'$;
2. Call algorithm 1 and algorithm 2, get the set of new activities Act_{new}, the relations set R_{pre} and R_{subs};
3. FOR $(a_i \in R_{pre}, i = 1; i <= length; i{+}{+})$ DO
4. $\eta(a_i, t_i)$;
5. IF $(t_k \in Act_{new}, l = 1; l <= count; l{+}{+})$ THEN
6. $P' \leftarrow P$;
7. $T' \leftarrow T' \cup \{t_k\}$;
8. $F' \leftarrow F' \cup \{t_i^{\bullet} \rightarrow t_k\}$;
9. $I' \leftarrow I' \cup \{I(t_k) = t^{\bullet} \otimes t_{i+1}^{\bullet}\}$;
10. FOR $(a_j \in R_{subs}, j = 1; j <= length; j{+}{+})$ DO
11. $\eta(a_i, t_i)$;
12. IF $(t_k \in Act_{new}, l = 1; l <= count; l{+}{+})$ THEN
13. IF $(a \in S_{pre}$ *and* $b \in S_{subs}$ *and* $a \dot{=}\dot{=} b)$ THEN
14. $P' \leftarrow P \cup \{P_j\}$;
15. $T' \leftarrow T' \cup \{t_k\}$;
16. $F' \leftarrow F' \cup \{t_k \rightarrow P_j\}$ and $\{t_k^{\bullet} \rightarrow t_j\}$;
17. $O' \leftarrow O' \cup \{O(t_k) = P_j \otimes t_i^{\bullet} \otimes t_{i+1}^{\bullet}\}$;
18. RETURN $LPN' = (P', T'; F', I', O', M')$;

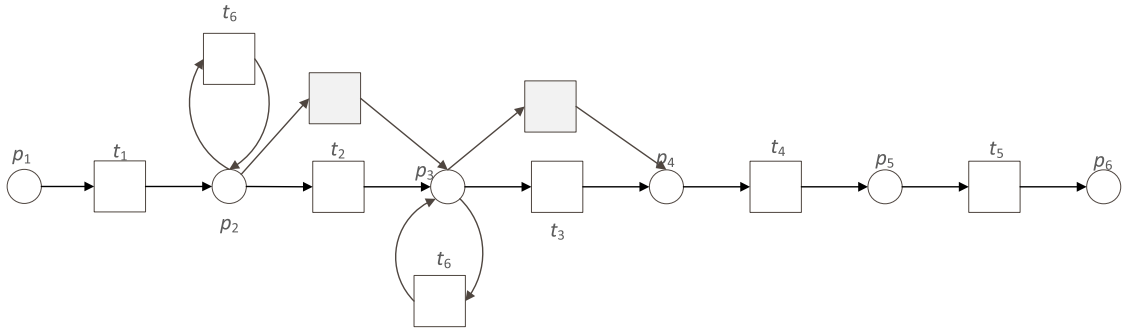

FIGURE 4. The repaired model RN_1 by the Fahland's approach.

TABLE 1. A comparison of added elements based on two repair approaches.

	Number of added places $\|P\|$	Number of added flow relation $\|F\|$	Number of added transitions $\|T+\tau\|$	Number of added repeated transitions $\|T\|$
The repaired model by our approach	1	5	1	0
The repaired model by the Fahland's approach	0	8	4	1

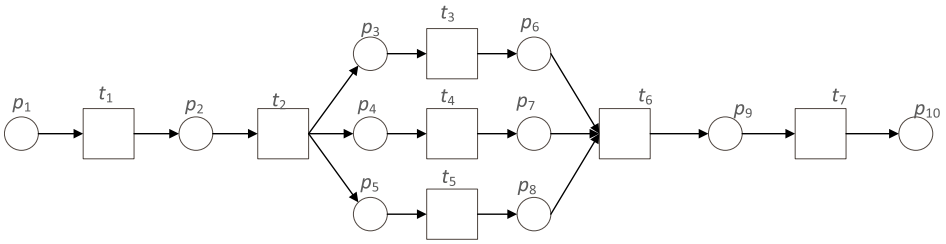

FIGURE 5. A Petri nets N_2 with a concurrent structure.

activities have a causal relation, the input logic function can be constructed. The logic relation between post-set places corresponding to transitions of precursor-set activities in the function is "or". Similarly, we collect the post-set activities of new activities and build a successor set, and then the relation of successor-set activities in the model is determined. When the transitions corresponding to successor-set activities in the model are a casual relation, the logic output function can be constructed. The logic relation between the pre-set places of transitions corresponding to successor-set activities in the function is "or".

V. THE LOGIC REPAIR OF THE MODELS WITH THE CONCURRENT RELATION

This section gives the definitions and algorithms for repairing models with a concurrent structure.

In the concurrent structure of the Petri net, all branches have the same start and end transitions. Suppose that t_s and t_e represent the start and end transitions of the concurrent structure, respectively. A Petri net can be expressed by a unique process tree to record the structures of a Petri net and its transitions.

In Example 2, Figure 5 shows a Petri nets model N_2 with a concurrent structure. An event log is $L_2 = \{< t_1, t_2, t_3, t_4, t_6, t_7 >, < t_1, t_2, t_4, t_3, t_6, t_7 >, < t_1, t_2, t_5, t_6, t_7 >\}$.

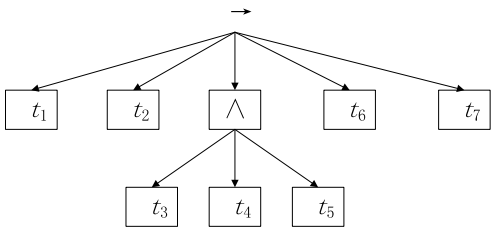

FIGURE 6. The process tree PT_1 of N_2.

The process tree of N_2denoted by PT_1 is shown in Figure 6.

$$N_2 = \rightarrow (t_1, t_2, \wedge(t_3, t_4, t_5), t_6, t_7)$$

The node of the concurrent structure can be found by traversing the process tree, and all leaf nodes can be found.

Algorithm 5 Compute the Set of Concurrent Start-Stop Pairs *CCSPS* and the Set of Concurrent Transitions *CTS*

Input: Non-leaf node of process tree PT denoted by N_l, and $PN = (P, T; F, M)$.

Output: The set of concurrent start-stop pairs denoted by *CCSPS* and the set of concurrent transitions denoted by *CTS*

1. $CSPS \leftarrow \varnothing$, $CON \leftarrow \varnothing$;
2. IF (N_l != $\varnothing$ AND $N \in \oplus$) DO
3. IF (N_l="$\wedge$") THEN
4. $CSPS \leftarrow CSPS \cup \{{}^{\bullet}({}^{\bullet}(SN)), ((SN)^{\bullet})^{\bullet}\}$;
5. ELSE
6. FOR all the node $\in N$ DO
7. *CCSPS*(node, *PN*);
8. $CTS \leftarrow CTS \cup \{SN\}$;
9. RETURN *CSPS*, *CTS*;

Then combined with the Petri net, the start transition and end transitions of these leaf nodes can be found. If a process tree has a node with "$\wedge$", the corresponding process model has a concurrent structure. Then, the sub-nodes can be obtained, e.g., t_3, t_4 and t_5 are the sub-nodes of the node "$\wedge$". The two-tuple consists of the start and end transitions of the concurrent structure, which is called a concurrent start-stop pair. Then we can collect all concurrent start-stop pairs in the Petri net, and construct the set of concurrent start-stop pairs. The formal definition is as follows:

Definition 16 (Start-Stop Pairs): Let $A \in \mathcal{A}$ and $PN = (P, T; F, M)$. PT is the process tree of PN and the concurrent start-stop pair denoted by CSP is a tuple (t_s, t_e), where

(1) $\exists N \in PT$ and $N = \wedge$; and
(2) $t_s = {}^{\bullet}({}^{\bullet}(SN))$, $t_e = ((SN)^{\bullet})^{\bullet}$, and SN is the sub-node of N.

The set of concurrent start-stop pairs denoted by *CSPS* includes all concurrent start-stop pairs, i.e., $CSPS = \{(t_s, t_e) | t_s = {}^{\bullet}({}^{\bullet}(SN)), t_e = ((SN)^{\bullet})^{\bullet}, \forall N \in PT \text{ and } N = \wedge\}$.

Definition 17 (Concurrent Transitions): If there are two traces $\sigma, \sigma' \in L$ for $a, b \in \&(\sigma)$: $a > b$ and for $a, b \in \&(\sigma')$: $b > a$, then a and b are regarded as concurrent transitions.

Given the above definitions, the aim of the following algorithm is to identify the concurrent structure. The concurrent structure can be repaired based on logic Petri nets. If there are other structures on each branch of the concurrent structure, we treat each branch as a whole separately. The following is the algorithm to obtain the set of concurrent start-stop pairs and the set of concurrent transitions based on Petri nets.

The definitions of the projection and the sub-trace log are given below.

Definition 18 (The Projection From Traces to the Set of Transitions): Let TS be a transition set, and $B \subseteq TS$ be a subset of the set TS. For $\sigma_i \in B^*$, $\sigma_i|_B$ denotes the projection from σ_i to B.

Algorithm 6 Model Repair Algorithm Based on the Concurrent Relation

Input: **A** complete event log $L \in B(A^*)$, and a workflow net $WFN = (P, T; F, M, i, o)$

Output: Repaired model of the logic Petri net $LPN'' = (P'', T''; F'', I'', O'', M'')$

1. $N \leftarrow LPN''$, $L_{sub} \leftarrow \varnothing$;
2. Using Algorithm 5 to get the set of concurrent start-stop pairs *CCSPS* and the set of concurrent transitions *CTS*;
3. FOR ($\sigma_i \in L$, $i = 1$; $i <= length$; i++) DO
4. $\sigma_i|_{CTS}$;
5. $L_{sub} \leftarrow L_{sub} \cup \{\sigma_i|_{CTS}\}$;
6. FOR ($\sigma_i \in L_{sub}$, $i = 1$; $i <= count$; i++) DO
7. Using Algorithm 2 to get the relation between the transitions in the sub-trace denoted by R_{sub};
8. IF ($t_a || t_b$ and $t_a \in \sigma_m$ and $t_b \notin \sigma_m$) THEN
9. $CST_{con} \leftarrow CST_{con} \cup \{t_a, t_b\}$;
10. IF ($t_{CSTcon} \in CST_{con}$) THEN
11. $P_{CSTpcon} \leftarrow P_{CSTpcon} \cup \{t^{\bullet}_{CSTcon}\}$;
12. $O'' \leftarrow O'' \cup \{O(\pi_1(CSPS)_{\downarrow T}) = [(\wedge P_{CSTpcon})] \otimes P_{(CST-CSTpcon)}\}$;
13. $P_{CSTscon} \leftarrow P_{CSTscon} \cup \{t^{\bullet}_{CSTcon}\}$;
14. $I'' \leftarrow I'' \cup \{I(\pi_2(CSPS)_{\downarrow T}) = [\wedge(P_{CSTscon})] \otimes P_{(CST-CSTscon)}\}$;
15. RETURN $LPN'' = (P'', T''; F'', I'', O'', M'')$;

For example, $aabc|_{\{a,b\}} = aac$. The concept of the projection can also be applied on multiple sets, for example, $[a^3, b, c^2]|_{\{a,b\}} = [a^3, b]$.

Definition 19 (The Sub-Trace Log): Let TS be a set of transitions and $B \subseteq TS$ be a sub-set of the set TS. For the log $L \in \mathrm{B}(A^*)$, $\forall \sigma_m \in L$, and $\sigma = \{t_1, t_2, \ldots, t_n\}$, $\sigma_i|_B$ is the projection from each trace to the set of transitions. The resulting sub-trace $\sigma_i|_B$ is denoted by $L_{sub} = \{\sigma_i|_B, 1 \le i \le m\}$.

For example, in Example 2, $L_2 = \{< t_1, t_2, t_3, t_4, t_6, t_7 >, < t_1, t_2, t_4, t_3, t_6, t_7 >, < t_1, t_2, t_5, t_6, t_7 >\}$, and N_2 is a Petri net model with a concurrent structure. We have $CTS = \{t_3, t_4, t_5\}$, $\sigma_1|_{CTS} =< t_3, t_4 >$, $\sigma_2|_{CTS} =< t_4, t_3 >$, $\sigma_3|_{CTS} =< t_5 >$, and the sub-trace log is $L_{sub} = \{< t_3, t_4 >, < t_4, t_3 >, < t_5 >\}$.

After the set of transitions in a concurrent structure in the model is obtained, we then project each trace of the log to the set. Then, these sub-traces can be obtained.

After obtaining the sub-trace log, we can judge the concurrent relations in the log by Algorithm 2. The logic input and output functions can be constructed. The logic Petri net model repair algorithm based on the concurrent structure is given as follows.

According to [20], the logic relation between transitions can be obtained. For example, $L_2 = \{< t_1, t_2, t_3, t_4, t_6, t_7 >, < t_1, t_2, t_4, t_3, t_6, t_7 >, < t_1, t_2, t_5, t_6, t_7 >\}$ and N_2 is a Petri net model with a concurrent structure. We can get the concurrent start-stop pair $CSP = \{t_2, t_6\}$. The start and end transitions of the concurrent structure are t_2 and t_6, respectively. The set

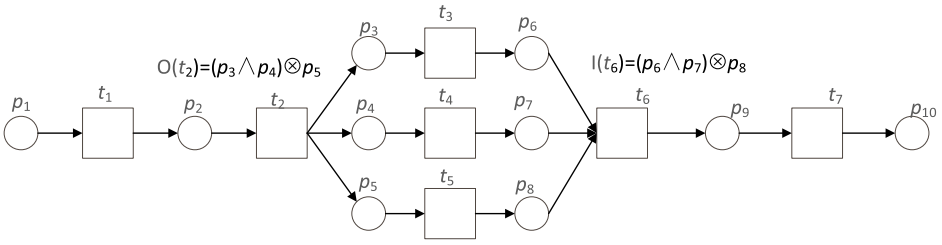

FIGURE 7. The repaired model by our approach.

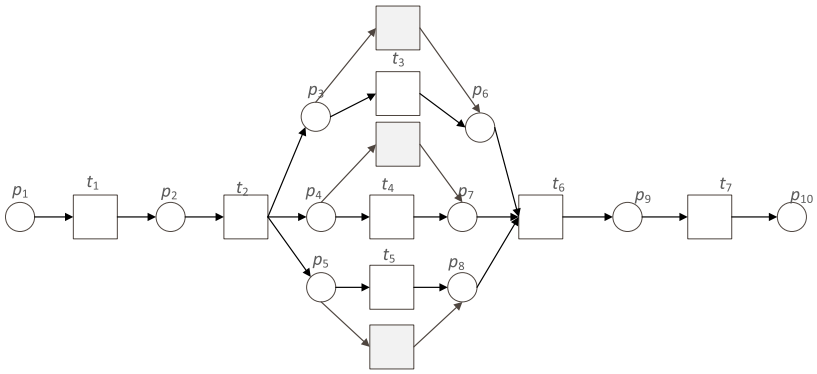

FIGURE 8. The repaired model by the Fahland's approach.

TABLE 2. A comparison of the added elements based on two repair approaches.

	Number of added places $\|P\|$	Number of added flow relation $\|F\|$	Number of added transitions $\|T+\tau\|$	Number of added repeated transitions $\|T\|$
The repaired model by our approach	0	0	0	0
The repaired model by the Fahland's approach	0	6	3	0

of concurrent transitions is $CTS=\{t_3, t_4, t_5\}$, $\sigma_1|_{CTS} =< t_3, t_4 >$, $\sigma_2|_{CTS} =< t_4, t_3 >$, $\sigma_3|_{CTS} =< t_5 >$, and the sub-trace log $L_{sub} = \{< t_3, t_4 >, < t_4, t_3 >, < t_5 >\}$. The set of concurrent relations is obtained by Algorithm 6, i.e., $CST_{con} = \{t_3, t_4\}$. The relation between pre-set places of transitions with concurrent relations is "and", and the relation between pre-set places of remaining transitions is "or". Here, the logic output function is $O(t_2) = (p_{3\wedge}p_4)\otimes p_5$, and the logic input function is $I(t_6) = (p_{6\wedge}p_7) \otimes p_8$. The repaired model is shown in Figure 7.

We can use the Fahland's approach to get the repaired model as shown in Figure 8. The obtained model by our approach is more simple and accurate than the repaired model by the Fahland's approach. Moreover, the precision of the obtained model by our approach is very high, and there are no different structures to describe the same behavior. At the same time, there will be no self-loop. Our approach can also solve the problem that the concurrent relations cannot be identified in the trace. Table 2 gives a comparison of the number of added elements based on two repair approaches in Example 2.

In fact, the approach proposed in [20] requires a large number of complex logic operations to obtain logical expressions. Besides, it needs to consider the frequency of activities, which makes the process of model repair very complex. Our paper proposes a new approach by identifying new activities to obtain the deviations between a model and a log. Then a precursor set and a successor set of the activities can be obtained. Based on the relations among the activities in each set, the logic relation functions can be constructed. Thus, our approach is very simple but more efficient.

The complexity of algorithm 6 determines the complexity of the other algorithms. Because this algorithm has gathered all the algorithms in front. The time complexity of algorithm 6 is O(n), where n is the length of the logs. As a result, our approach is very efficient.

This section presents the steps of the repair approach: the process tree is used to formalize the model, and to get concurrent start-stop pairs. We then find the concurrent structures in the original model and construct the set of concurrent transitions. Then, each trace in the log is projected to a set

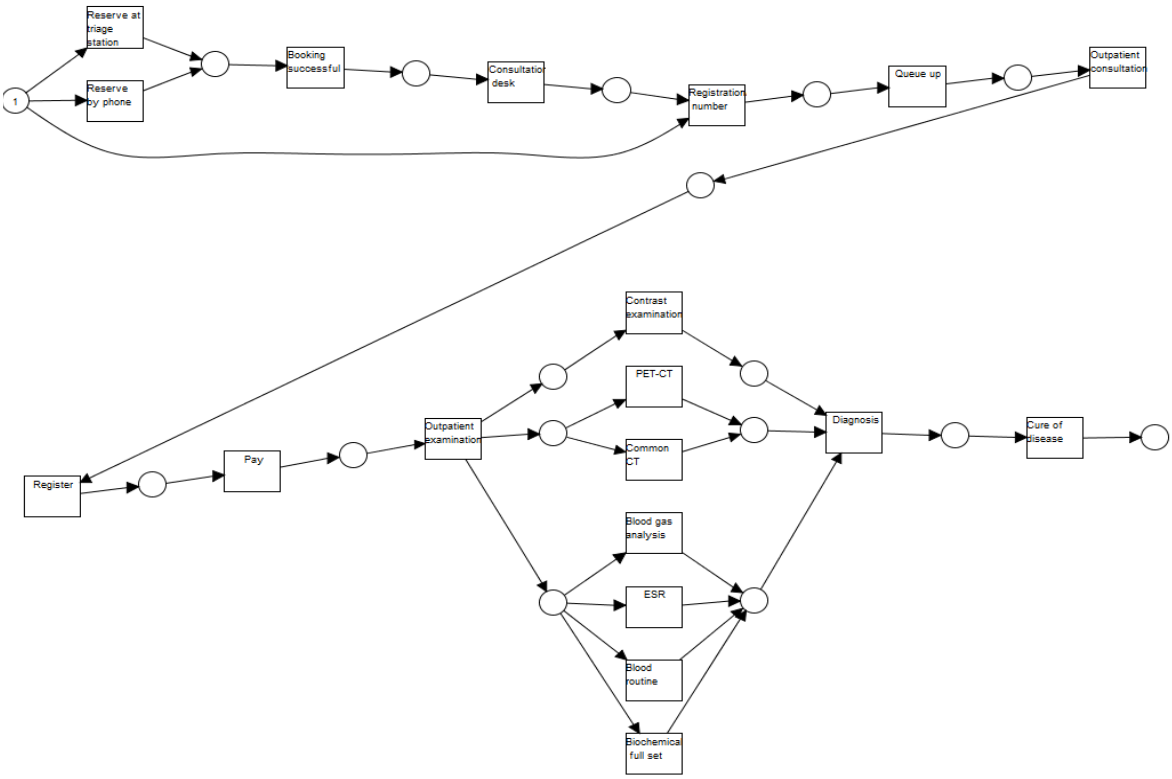

FIGURE 9. The process model for hospital outpatients.

of concurrent transitions, and we can thus obtain a sub-trace and form a sub-trace log. Based on the decision algorithm of the transition relation, the relations of transitions in the sub-trace log is determined. Then the logic output function can be constructed, and transitions of the concurrent relation can be collected. The logic relation between the pre-set places with concurrent relations is ''and'', and the logic relation between the remaining places is ''or''. Similarly, logic input functions can be constructed, and then transitions of concurrent relations can be collected. The logic relation between the post-set places where all transitions are concurrent relations is ''and'', and the logic relation between the remaining places is ''or''.

VI. SIMULATION EXPERIMENTS

This section will carry out experiments and compare two proposed approaches with the Fahland's approach. The process and event logs used in the experiment are derived from a hospital, and the event log can be accessible at: https://pan.baidu.com/s/1XY0-5jlSGY_Ga8BWhO4QVw. The sub-process-based repair approach is implemented in the Process Mining Toolkit ProM6.6, available from http://www.promtools.org/prom6/.

A. MODEL AND DATA FOR EXPERIMENTS

Taking a business process from a thoracic surgery in a hospital as an example, the process mining modeling is performed on the basis of the original event logs. Figure 9 shows the process model of hospital outpatients. The following describes main activities of the process: first of all, the patient makes an appointment in the hospital by two ways: telephone booking and triage reservation. If the booking is successfully done, a ''successful booking'' message is sent back to the patient. Then the patient can go to the information desk by consulting some related problems. Patients who book successfully can get a reservation number. Patients who fail to make an appointment can continue the reservation. The patient can also see the doctor without making an appointment. Hospital patients need to wait in line and go to outpatient consultation in order. In the process of inquiry, the doctor will register the patients' personal information and conditions, and then let them pay for it. The doctor will check what the patients need through the clinic examination. There are two types of the CT examination: PET-CT and common CT. There are four types of the clinical examination: biochemistry, blood gas analysis, blood routine and ESR, and there is another type of examination: the contrast examination. They will diagnose according to the results of the examination and then start the treatment.

After learning the process, the corresponding event log can be obtained. At first, the logs need to be filtered such that they deviate from the process model. For example, a process may lack a start activity or is not a complete one. Table 3 shows the main properties of these event logs, the total number of traces in the event log, the length of traces, and the total number of

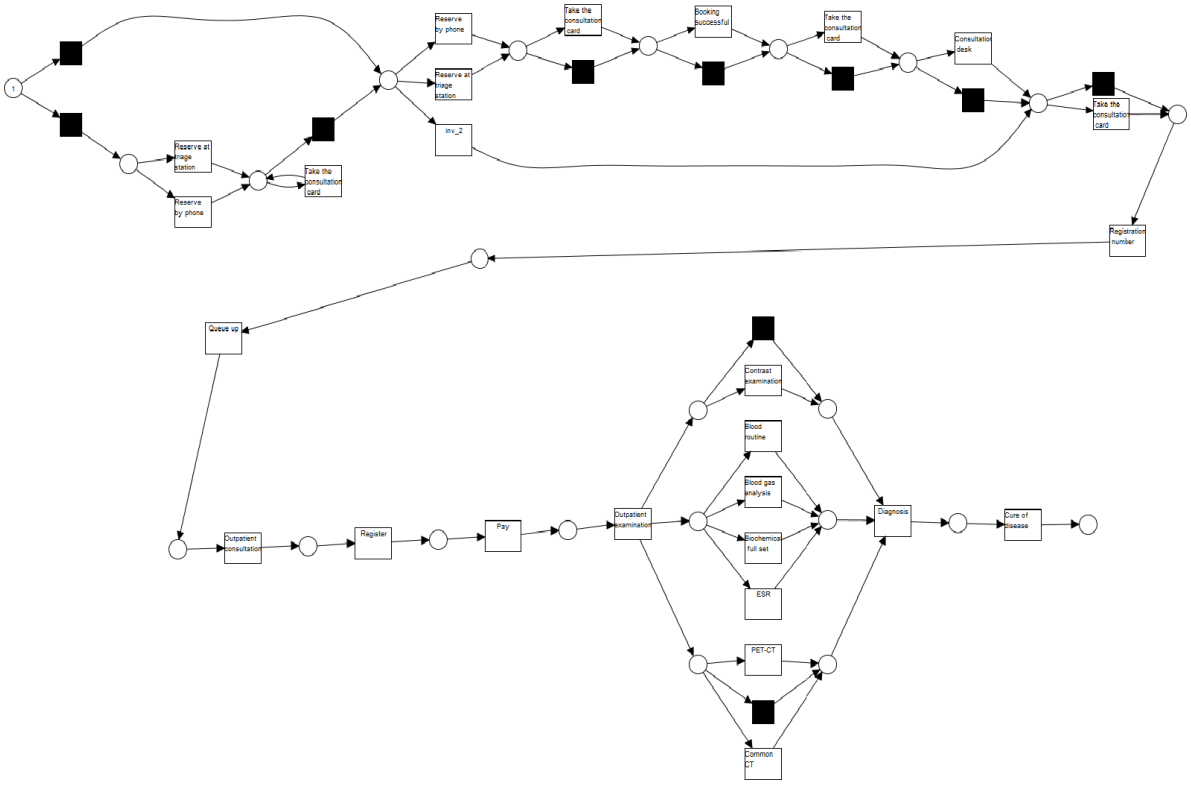

FIGURE 10. The repaired model for outpatients by the Fahland's approach.

TABLE 3. Three groups of detailed log information.

Log	Log		Total Deviation
	Trace number	Length	
L_1	1208	10—14	2674
L_2	1598	10—14	2965
L_3	1800	10—14	3247

deviations in the trace. The log L_1 contains 1208 traces with a total number of 2674 deviations, and the length of traces ranges from 10-14. There are 1598 traces in log L_2 with a total number of 2965 deviations, and the length of traces ranges from 10-14. The log L_3 contains 1800 traces with a total number of 3247 deviations, and the length of traces ranges from 10-14.

B. MODEL REPAIR EXPERIMENTS BASED ON LOGIC PETRI NETS

This paper compares and analyzes the repaired model by our proposed approach and the Fahland's approach. The idea of the Fahland's approach is to use the alignment to discover the deviation between transitions in the model and activities in the log. We can obtain unfitted sub-logs, and then mine sub-logs by an inductive mining algorithm. Finally, the sub-process is added as a self-loop to the appropriate places in the original model. Due to the addition of self-loops in the model, the sub-process can be repeated indefinitely in the repaired model. In the actual process model, this sub-process may not be allowed to repeat many times, which will generate a lot of unnecessary traces, and greatly reduce the accuracy of the model.

We carry out the experiment by using the model in Figure 9 and event logs in Table 2. The repaired model by the Fahland's approach is shown in Figure 10.

From Figure 10, we can see that compared with the original model, the repaired model adds 10 invisible transitions, 31 flow relations, and there are multiple repetitive activities, which greatly increases the complexity of the model. Besides, the repaired model greatly changes the structure of the original model.

The repaired model by our approach is shown in Figure 11. In the model, a new activity "make the consultation card" is inserted, and a place is added to control the successor transitions of the new activity. According to the event log and the original model, the logic input function of the new activity is $I(t_{20}) = p_2 \otimes p_3$, and the logic output function is $O(t_{20}) = p_4 \otimes p_{18}$. The logic input function of the "information consultation" activity is $I(t_4) = p_{18} \otimes p_3$. Similarly, according to the proposed mining approach, we can get the logic output function of the "outpatient examination" activity, i.e., $O(t_{10}) = (p_{10} \otimes p_{12}) \wedge p_{11}$, and the logic input function of the "diagnose" activity is $I(t_{10}) = (p_{13} \otimes p_{15}) \wedge p_{14}$.

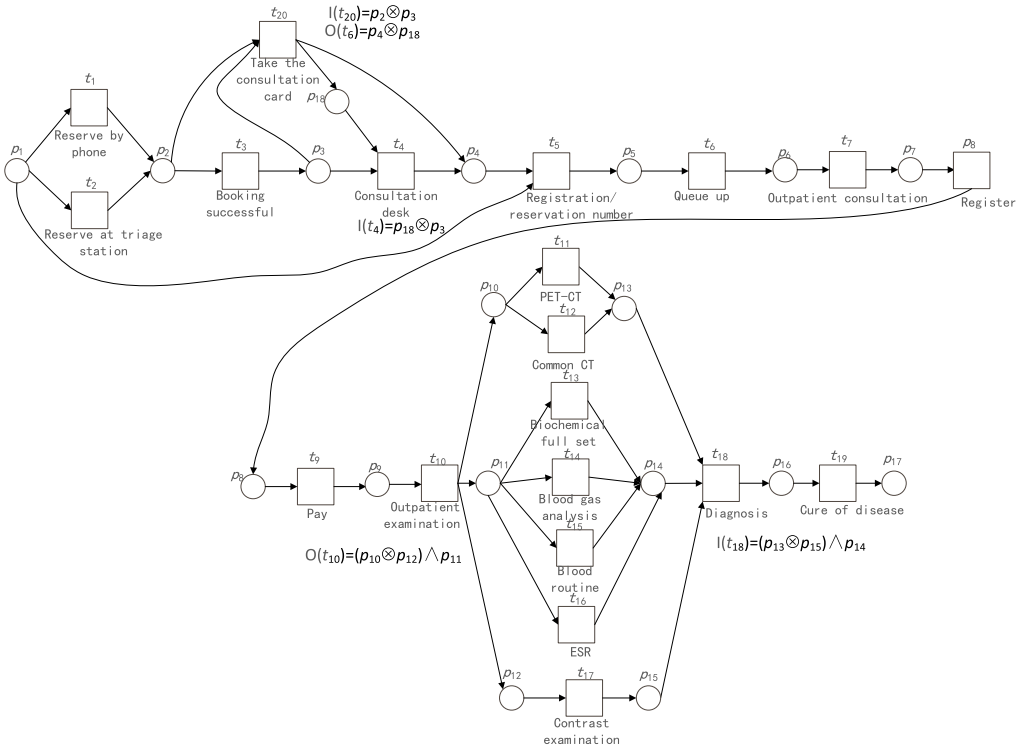

FIGURE 11. The repaired model for outpatients by our approach.

TABLE 4. The result by the Fahland's approach.

	Number of added places $\|P\|$	Number of added flow relation $\|F\|$	Number of added transitions $\|T+\tau\|$	Number of added repeated transitions $\|T\|$	Fitness *Fit*	Precision *Prec*
L_1	6	31	17	5	0.9624	0.8146
L_2	6	31	17	5	0.9580	0.8233
L_3	6	31	17	5	0.9658	0.8015

TABLE 5. The result by our approach.

	Number of added places $\|P\|$	Number of added flow relation $\|F\|$	Number of added transitions $\|T+\tau\|$	Number of added repeated transitions $\|T\|$	Fitness *Fit*	Precision *Prec*
L_1	1	5	1	0	0.9798	0.9258
L_2	1	5	1	0	0.9868	0.9387
L_3	1	5	1	0	0.9884	0.9426

From Figure 11, we can see that compared with the original model, the repaired model adds a new activity and 5 flow relations, and no invisible transitions and repetitive activities are added. Compared with the sub-process-based repair approach [4], our approach greatly improves the conciseness of the repaired model. There are two or more activities as logic input/output transitions. Its input and output control the repaired model, which makes the model repair in order, and ensure the precision of the model. The repaired model by our approach is very similar to the structure of the original model, and is regarded as a good repair approach.

Tables 4 and 5 respectively show the experimental results of two repair approaches, including the number of added places $|P|$, flow relations $|F|$, the total number of

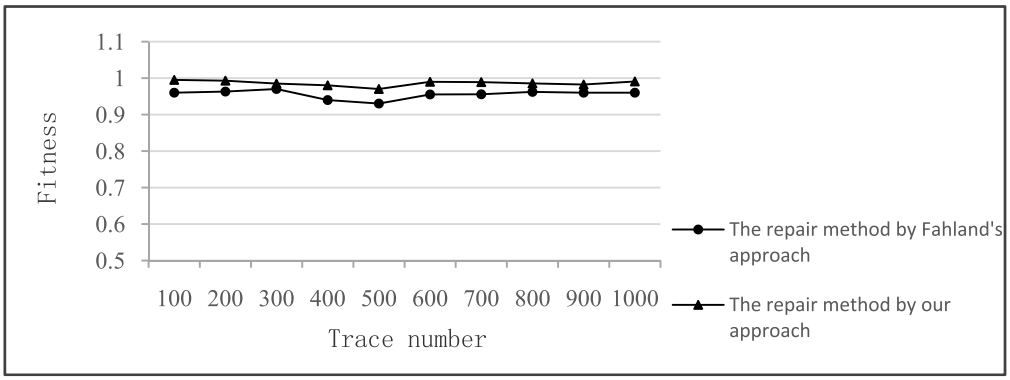

FIGURE 12. The change of fitness.

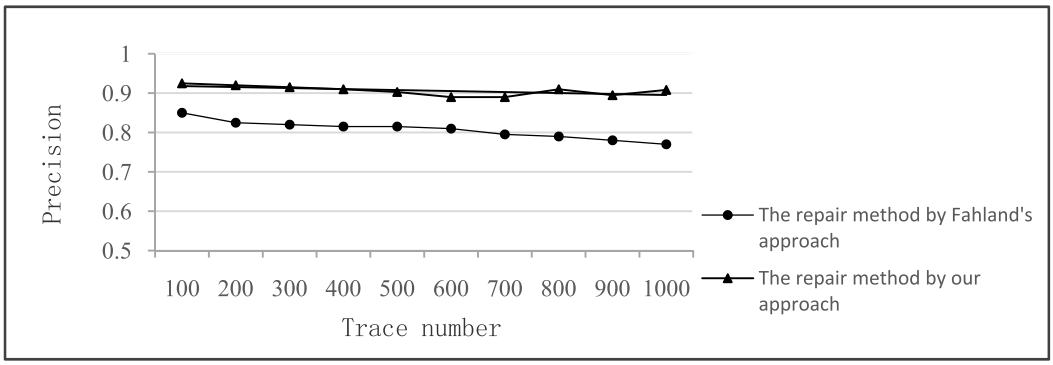

FIGURE 13. The change of precision.

transitions $|T + \tau|$, the number of repeated transitions $|T|$, and the fitness and precision values.

C. MODEL EVALUATION

The repaired model is used to compare with the original model. The Fahland's repair approach and our approach are compared. The variation of the fitness and the precision are shown in Figures 12 and 13. As the number of traces increases, the fitness of our approach is better than that of the Fahland's approach.

Figure 13 shows the precision obtained by the two repair approaches. We can see that the precision of the model by the Fahland's approach decreases with the increasing number of traces. Such effect is significantly lower than that by our approach. The precision of the repair approach by our approach keeps a relatively high level.

VII. CONCLUSIONS

This paper proposes a model repair approach based on the logic Petri net. There are no different structures describing the same behavior in the repaired model. By considering the logic relation between new activities and those activities in the original model, the repaired model can correctly describe the actual process. The structure of the repaired model is similar with that of the original one. The application of the logic Petri net repair approach is very effective to improve the fitness and the precision of the repaired model. In the future, we will propose some approaches to repair the process models containing other structures and relations based on logic Petri nets. We will use some other extended Petri nets [34]–[36] to conduct process mining in our future work.

REFERENCES

[1] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin, Germany: Springer, 2011.
[2] W. M. P. van der Aalst *et al.*, "Process mining manifesto," in *Business Process Management Workshops* (Lecture Notes in Business Information Processing). Berlin, Germany: Springer, 2012, pp. 169–194.
[3] C. Li, M. Reichert, and A. Wombacher, "Mining business process variants: Challenges, scenarios, algorithms," *Data Knowl. Eng.*, vol. 70, no. 5, pp. 409–434, May 2011.
[4] 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.
[5] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens, "Robust process discovery with artificial negative events," *J. Mach. Learn. Res.*, vol. 10, no. 9, pp. 1305–1340, 2009.
[6] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst, "Genetic process mining: An experimental evaluation," *Data Mining Knowl. Discovery*, vol. 14, no. 2, pp. 245–304, 2007.
[7] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, "Discovering expressive process models by clustering log traces," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 8, pp. 1010–1027, Aug. 2006.
[8] J. De Weerdt, S. K. L. M. van den Broucke, J. Vanthienen, and B. Baesens, "Leveraging process discovery with trace clustering and text mining for intelligent analysis of incident management processes," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2012, pp. 1–8.
[9] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," in *Applications and Theory of Petri Nets*. Berlin, Germany: Springer, 2011, pp. 387–412.

[10] R. P. J. C. Bose, W. M. P. van der Aalst, I. Zliobaite, and M. Pechenizkiy, "Dealing with concept drifts in process mining," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 154–171, Jan. 2014.
[11] W. M. P. van der Aalst, and C. Stahl, *Modeling Business Processes: A Petri Net-Oriented Approach*. Cambridge, MA, USA: MIT Press, 2011, pp. 13–75.
[12] L. Wen *et al.*, "Mining process models with prime invisible tasks," *Data Knowl. Eng.*, vol. 69, no. 10, pp. 999–1021, 2010.
[13] L. Wen *et al.*, "Mining process models with non-free-choice constructs," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 145–180, 2007.
[14] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin, Germany: Springer, 2011, pp. 64–162.
[15] 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.
[16] A. Rozinat and W. M. P. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Inf. Syst.*, vol. 33, no. 1, pp. 64–95, 2008.
[17] 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.
[18] Y. Sun, Y. Du, and M. Li, "A repair of workflow models based on mirroring matrices," *Int. J. Parallel Program.*, vol. 45, no. 4, pp. 1001–1020, 2016.
[19] Y. Du, C. Jiang, and M. Zhou, "A Petri net-based model for verification of obligations and accountability in cooperative systems," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 39, no. 2, pp. 299–308, Mar. 2009.
[20] Y. Y. Du *et al.*, "A method of process mining based on logic Petri nets," *Acta Electron. Sinica*, vol. 44, no. 11, pp. 2743–2751, 2016.
[21] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[22] C. Lin, H.-K. Yang, and Z.-G. Shan, "Application of Petri nets to bioinformatics," *Chin. J. Comput.*, vol. 30, no. 11, pp. 1889–1900, 2007.
[23] 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.
[24] Y. Du, L. Qi, and M. Zhou, "A vector matching method for analysing logic Petri nets," *Enterprise Inf. Syst.*, vol. 5, no. 4, pp. 449–468, 2011.
[25] 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.
[26] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. M. P. van der Aalst, "Alignment based precision checking," in *Business Process Management Workshops*. Berlin, Germany: Springer, 2013, pp. 137–149.
[27] L. Wang, Y. Du, and W. Liu, "Aligning observed and modelled behaviour based on workflow decomposition," *Enterprise Inf. Syst.*, vol. 11, no. 8, pp. 1207–1227, Jul. 2017.
[28] W. M. P. van der Aalst *et al.*, "Soundness of workflow nets: Classification, decidability, and analysis," *Formal Aspects Comput.*, vol. 23, no. 3, pp. 333–363, 2011.
[29] X. Lu, M. Zhou, A. C. Ammari, and J. Ji, "Hybrid Petri nets for modeling and analysis of microgrid systems," *IEEE/CAA J. Autom. Sinica*, vol. 3, no. 4, pp. 347–354, Oct. 2016.
[30] N. Ran, H. Su, and S. Wang, "An improved approach to test diagnosability of bounded Petri nets," *IEEE/CAA J. Autom. Sinica*, vol. 4, no. 2, pp. 297–303, Apr. 2017.
[31] 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.
[32] G. Liu, "Complexity of the deadlock problem for Petri nets modeling resource allocation systems," *Inf. Sci.*, vol. 363, pp. 190–197, 2016.
[33] G. Liu, "Some complexity results for the soundness problem of workflow nets," *IEEE Trans. Services Comput.*, vol. 7, no. 2, pp. 322–328, Apr./Jun. 2014.
[34] H. Li, J.-X. You, H.-C. Liu, and G. Tian, "Acquiring and sharing tacit knowledge based on interval 2-tuple linguistic assessments and extended fuzzy Petri nets," *Int. J. Uncertainty Fuzziness Knowl.-Based Syst.*, vol. 26, no. 1, pp. 43–65, 2018.
[35] H.-C. Liu, X. Luan, Z. Li, and J. Wu, "Linguistic Petri nets based on cloud model theory for knowledge representation and reasoning," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 717–728, Apr. 2018.
[36] H.-C. Liu, J.-X. You, Z. Li, and G. Tian, "Fuzzy Petri nets for knowledge representation and reasoning: A literature review," *Eng. Appl. Artif. Intell.*, vol. 60, pp. 45–56, Apr. 2017.

XIZE ZHANG received the B.S. degree from the Shandong University of Science and Technology, Qingdao, China, in 2016, where he is currently pursuing the M.S. degree with the College of Computer Science and Engineering. His 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 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 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 are 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 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 20 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 (ICNSC'2018).

HAICHUN SUN received the B.S. and M.S. degrees in computer science from the Shandong University of Science and Technology, Qingdao, China, in 2007 and 2010, respectively, and the Ph.D. degree in computer software and theory from Tongji University, Shanghai, China, in 2015. She is currently an Assistant Professor with the College of Information Technology and Network Security, People's Public Security University of China, Beijing, China. She has published over 10 papers on journals and conferences, such as the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, and the WISE 2014 International Conference. Her current research interests include information service, Petri nets, and service-oriented computing. She is a member of the Professional Committee of the Internet Information Service of Chinese Association of Automation.