Computing and Informatics, Vol. 39, 2020, 264–297, doi: 10.31577/cai_2020_1-2_264

A LOGIC PETRI NET-BASED REPAIR METHOD OF PROCESS MODELS WITH INCOMPLETE CHOICE AND CONCURRENT STRUCTURES

Yuanxiu Teng, Liang Qi*, Yuyue Du

The College of Computer Science and Engineering
Shandong University of Science and Technology, Qingdao 266590, China
e-mail: 392828580@qq.com, {qiliangsdkd, yydu001}@163.com

Abstract. Current model repair methods cannot repair incomplete choice and concurrent structures precisely and simply. This paper presents a repair method of process models with incomplete choice and concurrent structures via logic Petri nets. The relation sets are constructed based on process trees, including branch sets, choice activity sets and concurrent activity sets. The deviations are determined by analyzing the relation between relation sets and activities in the optimal alignment. The model repair method is proposed for models with incomplete choice and concurrent structures via logic Petri nets according to different deviation positions. Finally, the correctness and effectiveness of the logic Petri net-based repair method are illustrated by simulation experiments.

Keywords: Process model, model repair, process tree, alignment, logic Petri net

1 INTRODUCTION

Process mining builds a bridge between data mining and process modeling and analysis. It extracts effective information from the data and resources of the real business process system, and builds the process model based on different algorithms according to the required information. Process mining is widely used in the design, analysis, implementation and adjustment of the system process [1]. The three types of applications for process mining is process discovery, conformance checking, and process improvement. The process discovery algorithm is a function that maps the

* Corresponding author

event log to a process model, which can be a BPMN [2], YAWL [3], Petri net [4, 5] and so on. α algorithm takes an event log as the input, finds out the possible causal dependence according to the sequence of activities, and outputs a Petri net with the initial identification [6]. Heuristic mining builds models with the use of representation preference and frequency of causal networks. The basic idea of heuristic mining algorithm is that infrequent paths should not be included in the model [7]. Conformance checking is used to compare the behaviors of process models with the behaviors recorded in the event logs, and it looks for their commonality and heterogeneity, so as to ensure that the information system and the actual business process keep a good compliance. The classic conformance checking contains token replay and alignments [8]. Besides, the method proposed by [9] projects both systems and system models or logs onto sub-sets of activities to determine their performance, and is applicable to both log–model and model–model conformance checking. The literature [10] can analyze and classify deviations with respect to the intended purpose of data, and provides an algorithm to identify wide range of deviations.

Conformance checking can also be used to improve business processes, repair models, and evaluate process discovery algorithms [11]. When the process model and event logs do not match on the process, the process model needs to be repaired, which is the process improvement. The aim of process improvement is to make the process model better reflect the real business system and improve the performance of the model. Fitness, simplicity, precision, and generalization are four main types of model performance, and those four performances are used to evaluate the quality of process models. A new genetic process mining algorithm is proposed to discover a process model from event logs, and it uses the tree representation to ensure the soundness of the model [12]. To improve the quality of process models, many approaches of process improvement are proposed. The Fahland's approach uses alignments to align the runs of the given process models to the traces in the logs [13]. It mines loops that can replay sub-logs of non-fitting sub-traces. The Goldratt's approach and Knapsack's approach are proposed to improve the correspondence between a model and event logs, and speed up the repair [14]. The work in [15] presents a judgment to mine the sub-process as the branch of choice structures, instead of inserting the sub-process directly into the original model.

Logic Petri net is the further abstraction and extension of the Petri net with inhibitor arcs [16, 17]. It is more concise and can describe a large number of logic relations among complex activities. From the perspective of analyzing business process operation and resources, logic Petri nets can better analyze batch processing function and the uncertainty of activity enablement of business process systems. The work in [16] proposes a vector matching method, and analyzes the reachability, liveness, conservativeness, and reversibility of logic Petri nets based on reachability trees and the state equations. The precursor and successor of activities in the traces are defined in [17], and an extended log-based ordering relationship is proposed. The work in [18] is based on alignments to repair unfitting transitions in concurrent blocks and generates a new branch containing new activities. In some real business processes, concurrency and choice exist at the same time, we call this structure is an

incomplete choice structure or incomplete concurrent structure. The current model repair methods based on Petri nets consider to improve the fitness of models, often ignore the precison and simplicity, and cannot correctly describe the logic relation among activities.

Therefore, we present a logic Petri net-based repair method. This work has the following contributions:

1. Relation sets are constructed based on process trees, including choice and concurrent activity sets, and branch sets. These relation sets can precisely locate deviations for concurrent and choice structures combining with optimal alignments.
2. The algorithms of determining deviations are presented based on relation sets and event logs. The model repair method is proposed for models with incomplete choice and concurrent structures via logic Petri nets.
3. Experimental results illustrate the correctness and effectiveness of the repair method presented in the paper.

The rest of the paper is organized as follows. The background in relevant fields is introduced in Section 2. Section 3 presents an approach to repair models with incomplete choice structures. The method of repairing models with incomplete concurrent structures is proposed in Section 4. The results and performance analysis of simulation experiments are given in Section 5. Section 6 concludes our work and draws the future work.

2 PRELIMINARIES

This section introduces some basic notions, mainly including Petri nets [4], logic Petri nets [16], alignments [11], process trees [14], and the precursor and successor [17]. In the following content, $\mathcal{N}$ represents a natural number set, i.e., $\mathcal{N} = \{0, 1, 2, \dots\}$.

Definition 1 (Trace, event log)**.** Let $A \subseteq \mathcal{A}$ be a set, and $\mathcal{A}$ is all sets of activities. $\sigma \in A^*$ is called a trace that denotes a sequence of activities. An event log is a multi-set of traces denoted as $L \in \mathcal{B}(A^*)$.

Definition 2 (Tuple)**.** Let A be a set and a tuple with n elements is denoted by $r = (b_1, b_2, \dots, b_n) \in A \times A \times \cdots \times A$. The i^{th} element of r is denoted as $\pi_i(r)$.

For example, there is a tuple $r = (x, y, z) \in A \times A \times A$ with 3 elements, $\pi_1(r) = x$, $\pi_2(r) = y$, and $\pi_3(r) = z$.

Definition 3 (Pre-set, post-set)**.** Let $N = (P, T; F)$ be a net. P denotes a finite set of places, T denotes a finite set of transitions. $F \subseteq (P \times T) \cup (T \times P)$ denotes a finite set of directed arcs with each one from p to t or from t to p, where $p \in P$

and $t \in T$. x is a node in N and $\forall x \in P \cup T$, we have

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

where $^{\bullet}x$ and $x^{\bullet}$ represent the pre-set and post-set of x, respectively.

Definition 4 (Petri net)**.** A four-tuple $PN = (P, T; F, M)$ is a Petri net, where

1. $N = (P, T; F)$ is a net;
2. $M : P \to \mathcal{N}$ is a marking, $M(p)$ denotes the number of tokens in p, where $p \in P$; and
3. PN has the following transition firing rules:
   (a) for $t \in T$, if $\forall p \in^{\bullet} t : M(p) \geq 1$, then t is enabled under M, denoted as $M[t >$; and
   (b) if $M[t >$, then t can fire, and a new marking M' is generated, denoted as $M[t > M'$, and for $\forall p \in P$, we have

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

Definition 5 (Logic Petri net)**.** A six-tuple $LPN = (P, T; F, I, O, M)$ is called a logic Petri net, where

1. P denotes a finite set of places;
2. $T = T_D \cup T_I \cup T_O$ denotes a finite set of transitions, and $T \cap P = \phi$, for $\forall t \in T$, $^{\bullet}t \cap t^{\bullet} = \phi$, where
   (a) T_D is a set of classic transitions as in a Petri net;
   (b) T_I is a set of logic input transitions, for $\forall t \in T_I$, the input place of t is restricted by a logic input function $f_I(t)$; and
   (c) T_O is a set of logic output transitions, for $\forall t \in T_O$, the output place of t is restricted by a logic output function $f_O(t)$;
3. $F \subseteq (P \times T) \cup (T \times P)$ denotes a set of directed arcs with each one from p to t or from t to p, where $p \in P$ and $t \in T$;
4. I denotes a logic input function of t, where $t \in T_I$, and for $\forall t \in T_I$, $I(t) = f_I(t)$;
5. O denotes a logic output function of t, where $t \in T_O$, and for $\forall t \in T_O$, $O(t) = f_O(t)$;
6. $M : P \to \mathcal{N}$ is a marking, $M(p)$ denotes the number of tokens in p, where $p \in P$; and
7. LPN has the following transition firing rules:

(a) for $\forall t \in T_D$, the transition firing rule is consistent with that of Petri nets;
(b) for $\forall t \in T_I$, $I(t) = f_I(t)$. If $f_I(t)|_M =_\bullet T_\bullet$, then t can fire and is denoted as $M[t > M'$, and for $\forall p \in^\bullet t$, $M(p) = 1$, $M'(p) = 0$; for $\forall p \in t^\bullet$, $M(p) = 0$, $M'(p) = 1$; and for $\forall p \notin^\bullet t \cup t^\bullet$, $M'(p) = M(p)$; and
(c) for $\forall t \in T_O$, $O(t) = f_O(t)$. If $f_O(t)|_M =_\bullet T_\bullet$, then t can fire and is denoted as $M[t > M'$, and $\forall p \in^\bullet t$, $M'(p) = 0$; for $\forall p \in t^\bullet$, $M'(p) = 1$; and for $\forall p \notin^\bullet t \cup t^\bullet$, $M'(p) = M(p)$.

8. There are three symbols of the logic function: $\bigotimes$, $\wedge$ and $\vee$. $p_1 \bigotimes p_2 \cdots \bigotimes p_n$ denotes only one of $p_1 - p_n$ contains tokens; $p_1 \wedge p_2 \cdots \wedge p_n$ denotes each of $p_1 - p_n$ contains tokens; $p_1 \vee p_2 \cdots \vee p_n$ denotes at least one of $p_1 - p_n$ contains tokens; where $n \geq 2$.

For example, a logic Petri net denoted by LPN_1 is presented in Figure 1, where t_1 and t_3 are two logic transitions and t_2 is a classic transition. t_1 is a logic input transition, and $I(t_1) = p_2 \wedge (p_1 \bigotimes p_3)$. If t_1 fires, p_1 and p_3 cannot contain tokens at the same time, and $f_I(t_1) = p_2 \wedge (p_1 \bigotimes p_3) =_\bullet T_\bullet$, there will be two cases:

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

Besides, t_3 is a logic output transition, and $O(t_3) = p_6 \vee p_7$. When t_3 fires, its logic output function needs to satisfy $f_O(t_3) = p_6 \vee p_7 =_\bullet T_\bullet$, there are three cases:

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

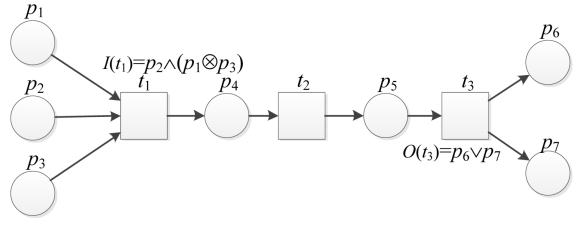


Figure 1. A logic Petri net model LPN_1

Definition 6 (Alignment)**.** Let $\sigma \in A^*$, and $PN = (P, T; F, M)$. A move is a pair $(a, t) \in (A \cup >>) \times (T \cup >>)$, where $>>$ denotes no move. A move sequence denoted by $\gamma = ((a_1, t_1)(a_2, t_2) \ldots (a_{|\gamma|}, t_{|\gamma|}))$ is called an alignment between σ and PN, where

1. $\pi_1(\gamma) = \sigma$ denotes a trace sequence generated by γ (ignoring $>>$); and

2. $m_i[\pi_2(\gamma) > m_f$ denotes a complete firing sequence generated by γ (ignoring $>>$); and
3. for each move (a, t), it is called a log activity if $a \in A$ and $t =>>$; it is called a model activity if $a =>>$ and $t \in T$; it is called a synchronous activity if $a \in A$ and $t \in T$; it is called an illegal activity otherwise.

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

Definition 7 (Optimal alignment)**.** Let $\sigma \in A^*$ and $PN = (P, T; F, M)$. $\gamma' \in \Gamma_{\sigma,PN}$ denotes an optimal alignment between σ and PN, if for $\forall\gamma' \in \Gamma_{\sigma,PN}$, we have $\sum_{(a,t)\in\gamma} lc(a,t) \leq \sum_{(a',t')\in\gamma} lc(a',t')$, where $lc(a,t)$ denotes a likelihood cost function. For each move $(a,\ t)$, if $a \in A$ and $t =>>$, $lc(a,t) = 1$; if $a =>>$ and $t \in T$, $lc(a,t) = 1$; if $a \in A$ and $t \in T$, $lc(a,t) = 0$. $\Gamma_{\sigma,PN,lc}$ denotes a set of all optimal alignments between σ and PN.

Definition 8 (Process tree)**.** Let $A \in \mathcal{A}$ be a set of activities. The notation of $\bigoplus$ denotes an operator set, and $\bigoplus = \{\times, \rightarrow, \bigodot, \wedge\}$, where

1. $a \in A \cup \{\tau\}$ denotes a process tree, and τ is an invisible transition; and
2. if $PT_1, PT_2, \ldots, PT_n (n > 0)$ are process trees, and then $\bigoplus(PT_1, \ldots, PT_n)$ is also a process tree; $\times$ represents the choice relation among $PT_1, \ldots, PT_n$; $\rightarrow$ represents the sequential execution of $PT_1, \ldots, PT_n$; $\bigodot$ represents the loop structure of $PT_1, \ldots, PT_n$; and $\wedge$ represents the parallel structure of $PT_1, \ldots, PT_n$.

Definition 9 (Precursor, successor)**.** Let $L \in B(A^*)$ be a log where $A \in \mathcal{A}$. For $\forall\sigma \in L$, if an activity $a \in \delta(\sigma)$ and the position index of a in σ is i, the precursor of a is denoted as ${}^{\triangle}a$ at the position with index $i-1$; and the successor of a is denoted as $a^{\triangle}$ at the position with index $i+1$.

For example, there is a trace $\langle t_1, t_2, t_5, t_3, t_4, t_5\rangle$, t_5 is the precursor of t_3, i.e., ${}^{\triangle}t_3 = t_5$; t_4 is the successor of t_3, i.e., $t_3^{\triangle} = t_4$.

3 INCOMPLETE CHOICE STRUCTURES DEVIATION REPAIR

This section proposes a repair method of process models with incomplete choice structures. For incomplete choice structures, we first present choice relation sets based on process trees of process models, including head-to-tail places, branch sets and choice activity sets. By comparing and analyzing the logic relation between activities in traces and transitions of choice relation sets, a model repair method via logic Petri nets is proposed.

In the following, we use $PN = (P, T; F, M)$ to denote a four-tuple Petri net, and use PT to denote a process tree of PN.

Definition 10 (Tree relation)**.** Let $|\ \rightarrow$ be a relation symbol of PT, and $(t_i \cup \bigoplus)^*| \rightarrow (t_j \cup \bigoplus)^*| \rightarrow \ldots | \rightarrow (t_n \cup \bigoplus)^*$ is a tree relation, where $t_i, t_j, \ldots, t_n \in T$ and $\bigoplus = \{\times, \rightarrow, \bigodot, \wedge\}$.

Definition 11 (Node relation)**.** Let $n \in (T \cup \bigoplus)$ be a node of PT, $\bigoplus = \{\times, \rightarrow, \odot, \wedge\}$. The top layer of PT is called the root node, there are six node relation:

1. if $m| \rightarrow n$ and $m \in (T \cup \bigoplus)$, then m is called the parent node of n, denoted as $n.parent = m$;
2. if $n| \rightarrow m$ and $m \in (T \cup \bigoplus)$, then m is called the child node of n, denoted as $n.child = m$;
3. if $m| \rightarrow n$, $m| \rightarrow l$, $m| \rightarrow r$ and $m, l, r \in (T \cup \bigoplus)$, then l and r are called the left and right sibling nodes of n, respectively, where l and r are on the left and right sides of n, denoted as $n.lsib = l$ and $n.rsib = r$;
4. if $\exists n.child \in (T \cup \bigoplus)$ and $|n.child| > 1$, then $n.child.p_i$ denotes the i^{th} child node of n, where $i \in [1, |n.child|]$;
5. if $n| \rightarrow \ldots | \rightarrow m$, $m.child =$ null, and $m.lib =$ null, then m is called the leftmost leaf node of n, denoted as $n.child.lp = m$; and
6. if $n| \rightarrow \ldots | \rightarrow m$, $m.child =$ null, and $m.rib =$ null, then m is called the rightmost leaf node of n, denoted as $n.child.rp = m$.

For example, a Petri net model denoted by PN_1 is presented in Figure 2, and PN_1 contains choice structures. Let $\sigma_1 = \langle a, t_1, t_5, t_2, t_4, b\rangle$, $\sigma_2 = \langle a, t_6, t_7, c, b\rangle$ be two traces. PN_1 has a choice structure with transitions t_1, t_2, t_3, t_4, t_5, t_6, t_7. In the trace σ_1, we have that t_1, t_2, t_4 and t_5 are concurrently enabled. It shows that the branch containing t_1, t_2, t_4 can occur concurrently with the branch containing t_5. In the original model PN_1, either of these two branches can be selected to occur. So there is an incomplete choice structure.

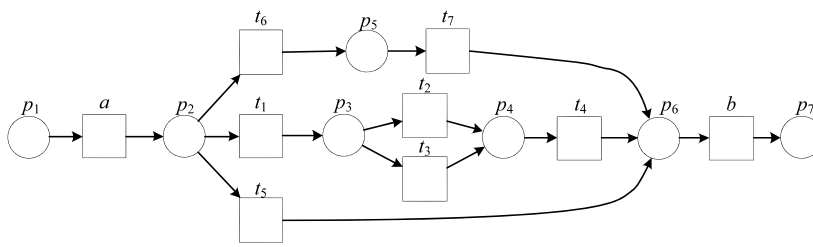


Figure 2. A Petri net model PN_1

Figure 3 shows the process tree of PN_1 represented by PT_1. $PT_1 = \rightarrow (a, \times((\rightarrow (t_1, \times(t_2, t_3), t_4), t_5, \rightarrow (t_6, t_7)), b)$. For PT_1, "$\rightarrow$"$| \rightarrow$ "$\times$", "$\times$"$| \rightarrow$ "$\rightarrow$", and "$\rightarrow$"$| \rightarrow t_1$ are three tree relations of PT_1. The root node is "$\rightarrow$". If $n =$ "$\times$", then $n.parent =$ "$\rightarrow$", $n.lsib = a$, and $n.rsib = b$. It can be seen that t_5 is the child node of n, $n.child.p_1 =$ "$\rightarrow$", $n.child.p_2 = t_5$, and $n.child.p_3 =$ "$\rightarrow$". Besides, $n.child.lp = t_1$, and $n.child.rp = t_7$.

Definition 12 (Head-to-tail place)**.** $[SF_P, SL_P]$ is called the head-to-tail place of a choice structure, where SF_P and SL_P are called the head place and the tail place, respectively. Let $n =$ "$\times$" be a node of PT, and it needs to satisfy:

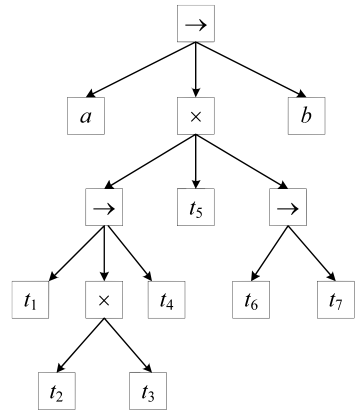


Figure 3. The process tree PT_1 of PN_1

1. if $n.child.p_i = m$, $m.child =$ null, and $0 < i \leq |n.child|$, then $SF_P =^{\bullet} m$ and $SL_P = m^{\bullet}$; and
2. if $n.child.p_i = m$, $\exists m.child \in (T \cup \bigoplus)$, $\bigoplus = \{\times, \rightarrow, \odot, \wedge\}$ and $0 < i \leq |n.child|$, then $SF_P =^{\bullet} (n.child.lp)$ and $SL_P = (n.child.rp)^{\bullet}$.

Definition 13 (Branch set)**.** Let n be a node of PT, $n =$ "$\times$" or $n =$ "$\wedge$", and a branch is defined as follows:

1. If $n.child = m$, $m.child =$ null, i.e., $n.child.child =$ null, then $\{m\}$ is a branch; and
2. if $n.child = m$, $\exists m.child.p_1, m.child.p_2, \ldots, m.child.p_k$, and $0 < k \leq |m.child|$, where
   (a) if $m =$ "$\rightarrow$", then $\{m.child.p_1,\ m.child.p_2, \ldots, m.child.p_k\}$ is a branch;
   (b) if $m =$ "$\times$", then $\{m.child.p_1\}$, $\{m.child.p_2\}$, $\ldots$, $\{m.child.p_k\}$ are each a branch; and
   (c) if $m =$ "$\wedge$", then $\{m.child.p_1\}$, $\{m.child.p_2\}$, $\ldots$, $\{m.child.p_k\}$ are each a branch.

B_S denotes a branch set that contains all branches of choice and concurrent structures.

Theorem 1. For $n =$ "$\times$", if $\exists n.child = m$ and $m.child =$ null, then $\{m\}$ is a branch.

Proof. If $\exists n.child = m$ and $m.child =$ null, i.e., $\exists n.child.child =$ null, it means that the child node of n is a leaf node. $\neg\exists m.child| \rightarrow \bigoplus$, where $\bigoplus = \{\times, \rightarrow, \odot, \wedge\}$, i.e., there are no structures behind m until SL_P. We have $SL_P = m^{\bullet}$. Since $m.parent = n$ and $n =$ "$\times$", n is the initial operator notation of the choice structure, then $SF_P =^{\bullet} m$. Thus, $\{m\}$ is a branch.

If $\{m\}$ is a branch and $n =$ "$\times$", it means $m^{\bullet} = SL_P$ and ${}^{\bullet}m = SF_P$. We have $\neg\exists m.child| \rightarrow \bigoplus$ and "$\times$"$| \rightarrow m$, i.e., $m = n.child$ and $m.child =$ null. □

Theorem 1 indicates that for the node of the process tree, if the node is "$\times$" and the child node of the node is a leaf node, then the child node alone is a branch of the choice structure.

Theorem 2. For $n =$ "$\times$" or $n =$ "$\wedge$", $m = n.child$, $0 < k \leq |m.child|$, $\{m.child.p_1\}$, $\{m.child.p_2\}$, ..., $\{m.child.p_k\}$ are each a branch.

Proof. For $n =$ "$\times$" or $n =$ "$\wedge$", there is $\exists q = m.lsib$ or $\exists q{=}m.rsib$, and $n| \rightarrow m.lsib$ or $n| \rightarrow m.rsib$. If $q.child.lp$ and m are contained in a branch, we have $M[q.child.lp >$, $\neg M[m >$, $M[q.child.lp > M'$, $M'[m >$ or $M[m >$, $\neg M[q.child.lp >$, $M[m > M'$, $M'[q.child.lp >$. For $n =$ "$\times$", $\exists M$, $M[q.child.lp >$, $\neg M[m >$, when $M[q.child.lp > M'$, $\neg M'[m >$ or $M[m >$, $\neg M[q.child.lp >$, $M[m > M'$, $\neg M'[q.child.lp >$, i.e., $q.child.lp$ and m are not contained in a branch. For $n =$ "$\wedge$", $\exists M, M[q.child.lp >$, $M[m >$, $M[q.child.lp > M'$, $M'[m >$ or $M[m > M'$, $M'[q.child.lp >$, i.e., $q.child.lp$ and m are not contained in a branch. Thus, $\{m.child.p_1\}$, $\{m.child.p_2\}$, ..., $\{m.child.p_k\}$ are each a branch. □

By Theorem 2, for the node denoted by "$\times$" or "$\wedge$" of the process tree, those child nodes of this node are not in the same branch.

Definition 14 (Choice activity set)**.** The choice activity set contains first and last choice activity sets. FC_S and LC_S denote the first and the last choice activity sets, respectively. If $n =$ "$\times$" is a node of PT, we have:

1. if $n.child = m$, $m.child =$ null, then $FC_S = \{m\}$, and $LC_S = \{m\}$;
2. if $n| \rightarrow \ldots| \rightarrow m$, $m =$ "$\wedge$" or $m =$ "$\times$", $k = |m.child|$, $q = n.child.lp$ and $m = q.parent$, then $FC_S = \{m.child.p_1, m.child.p_2, \ldots, m.child.p_k\}$;
3. if $n| \rightarrow \ldots| \rightarrow m$, $m =$ "$\wedge$" or $m =$ "$\times$", $k = |m.child|$, $q = n.child.rp$ and $m = q.parent$, then $LC_S = \{m.child.p_1, m.child.p_2, \ldots, m.child.p_k\}$; and
4. if $n| \rightarrow \ldots| \rightarrow m$, $m =$ "$\rightarrow$", and $m.child.child =$ null, then $FC_S = \{m.child.lp\}$, and $LC_S = \{m.child.rp\}$.

The algorithms of calculating branch sets, head-to-tail place and choice activity sets are proposed in the following.

Algorithm 1 calculates the branch set. For each node n of the process tree, if n is "$\wedge$" or "$\times$", and the child node of n is a leaf node, then this child node of n is contained in a branch. If the child node of n is "$\rightarrow$", and those child nodes of "$\rightarrow$" are leaf nodes, then all those leaf nodes are contained in a branch. If the child node of "$\rightarrow$" is "$\times$", then each node of the child node of "$\times$" is contained in a branch.

Algorithm 2 calculates the head-to-tail place and the choice activity set. For each node n of the process tree, if n is "$\times$", its head place is the pre-set of the leftmost leaf node of n, and its tail place is the post-set of the rightmost leaf node

Algorithm 1 $Calculate - B_S(PT)$

Input: A process tree denoted by PT
Output: The branch set denoted by B_S

```
1: B_S ← φ; C_TB ← φ;
2: for each n ∈ PT do
3:     if n = "∧"||n = "×" then
4:         for (i = 1; i ≤ |n.child| ; i++) do
5:             m_i ← n.child.p_i;
6:             if m_i.child = null then
7:                 B_S ← B_S ∪ {m_i};
8:             end if
9:             if m_i = "→" and m_i.child.child = null then
10:                for (j = 1; j ≤ |m_i.child| ; j++) do
11:                    C_TB ← C_TB ∪ m_i.child.p_j;
12:                end for
13:            end if
14:            if m_i = "→" and m_i.child = "×" then
15:                for each m_i.child.child do
16:                    C_TB ← C_TB ∪ m_i.child.child;
17:                end for
18:            end if
19:            B_S ← B_S ∪ C_TB;
20:        end for
21:    end if
22: end for
23: return B_S
```

of n. If the child node is a leaf node, then it is both contained in the first and last choice activity set. If the child node g of n is "$\times$" or "$\wedge$", and g is the parent node of the leftmost of rightmost leaf node of n, then each child node of g is contained in the first or last choice activity set. If g is "$\rightarrow$" and the child node of g is a leaf node, then the leftmost leaf node of g in contained in the first choice activity set, and the rightmost leaf node of g in contained in the last choice activity set.

For PT_1, we can obtain its head-to-tail place, branch set and choice activity set. Its head-to-tail place is denoted by $[SF_P, SL_P] = [p_2, p_6]$, and its branch set is denoted by $B_S = \{\{t_1, t_2, t_4\}, \{t_1, t_3, t_4\}, \{t_5\}, \{t_6, t_7\}\}$. The first choice activity set is denoted by $FC_S = \{t_1, t_5, t_6\}$ and the last choice activity set is denoted by $LC_S = \{t_4, t_5, t_7\}$.

Theorem 3. $|B_S|_{max}$ is the maximum length of elements in B_S. For $n =$ "$\times$" or $n =$ "$\wedge$", $n| \rightarrow w| \rightarrow \ldots| \rightarrow m$, if $w =$ "$\rightarrow$", $\exists m =$ "$\rightarrow$", $x = |w.child|$ and $y = |m.child|$, then $|B_S|_{max} = x + y$, where $w.child.child =$ null and $m.child.child =$ null; If $\neg\exists m =$ "$\rightarrow$", $|B_S|_{max} = x + 1$.

Algorithm 2 $Calculate - FLC(PT)$

Input: A process tree denoted by PT

Output: The head-to-tail place denoted by $[SF_P, SL_P]$, the first choice activity set denoted as FC_S, and the last choice activity set denoted as LC_S

1: $SF_P \leftarrow \phi$; $SL_P \leftarrow \phi$; $FC_S \leftarrow \phi$; $LC_S \leftarrow \phi$; $u \leftarrow \phi$; $q_1 \leftarrow \phi$; $q_2 \leftarrow \phi$;
2: **for** each $n \in PT$ **do**
3: **if** $n =$ "$\times$" **then**
4: $u \leftarrow |n.child|$;
5: $m \leftarrow n.child.p_1$;
6: $k \leftarrow n.child.p_u$;
7: $SF_P \leftarrow SF_P \cup^{\bullet} (m.child.lp)$;
8: $SL_P \leftarrow SL_P \cup (k.child.rp)^{\bullet}$;
9: **for** $(i = 1; i \leq |n.child|\,; i++)$ **do**
10: $g_i \leftarrow n.child.p_i$;
11: **if** $g_i.child = null$ **then**
12: $FC_S \leftarrow FC_S \cup \{g_i.child\}$;
13: $LC_S \leftarrow LC_S \cup \{g_i.child\}$;
14: **end if**
15: $q_1 \leftarrow n.child.lp$;
16: $q_2 \leftarrow n.child.rp$;
17: **if** $g_i =$ "$\times$"$||g_i =$ "$\wedge$" and $(g_i = q_1.parent)$ **then**
18: **for** each $g_i.child$ **do**
19: $FC_S \leftarrow FC_S \cup g_i.child$;
20: **end for**
21: **end if**
22: **if** $g_i =$ "$\times$"$||g_i =$ "$\wedge$" and $(g_i = q_2.parent)$ **then**
23: **for** each $g_i.child$ **do**
24: $LC_S \leftarrow LC_S \cup g_i.child$;
25: **end for**
26: **end if**
27: **if** $g_i =$ "$\rightarrow$" and $g_i.child.child$ **then**
28: $FC_S \leftarrow FC_S \cup g_i.child.lp$;
29: $LC_S \leftarrow LC_S \cup g_i.child.rp$;
30: **end if**
31: **end for**
32: **end if**
33: **end for**
34: **return** $[SF_P, SL_P]$, FC_S, LC_S

Proof. For $n =$ "$\times$" or $n =$ "$\wedge$", $n| \rightarrow w| \rightarrow \ldots| \rightarrow m$, $w =$ "$\rightarrow$", if $\exists d_1 = w.child.p_i$, $d_1.child =$ null, $d_2 = w.child.p_j$, $d_2.child =$ null, where $i, j \in [1, |w.child|]$, it means that $M[d_1 >$, $\neg M[d_2 > M'$, $M[d_1 > M'$, $M'[d_2 >$, so the maximum length of the element of B_S only containing d_1 and d_2 is 2. So if $\exists d = w.child$, $d.child =$ null, $|d| = x$, where $0 < x \leq |w.child|$, then the length of the element in B_S containing d is x. If $\exists m =$ "$\rightarrow$", if $\exists e_1 = m.child.p_i$, $e_1.child =$ null, $e_2 = m.child.p_j$, $e_2.child =$ null, where $i, j \in [1, |m.child|]$, it means that $M[e_1 >$, $\neg M[e_2 > M'$, $M[e_1 > M'$, $M'[e_2 >$, in the same way, the length of the element in B_S only containing e_1 and e_2 is 2. So if $\exists e = m.child$, $e.child =$ null, $|e| = y$, where $0 < y \leq |m.child|$, then the maximum length of the element in B_S containing e is y, thus, $|B_S|_{max} = x + y$; If $\exists m =$ "$\rightarrow$", it means that $M[m.child.p_i >$, $M[m.child.p_j >$, $M[m.child.p_i > M'$, $M'[m.child.p_j >$ or $M[m.child.p_i >$, $M[m.child.p_j >$, $M[m.child.p_i > M'$, $\neg M'[m.child.p_j >$, $i \neq j$ and $0 < i, j \leq |m.child|$, so the maximum length of the element in B_S containing $m.child$ is 1, thus, $|B_S|_{max} = x + 1$. □

By Theorem 3, in the process tree, the maximum length of the branch of the choice or concurrent structure is the sum of the number of leaf nodes of the child node of "$\rightarrow$" and the number of nodes "$\times$" and "$\wedge$".

Deviations are determined by judging whether the corresponding transitions of log activities are in a branch. For those log activities, we can obtain the branches that need to be concurrent with other branches. New activities that need to be inserted into the original models can also be obtained from the optimal alignment. The algorithm for determining the deviation position is given in the following.

Algorithm 3 calculates the deviation position for incomplete choice structures. We obtain the positions of SF_P and SL_P in the optimal alignment, and collect new activities. For activities between SF_P and SL_P in the optimal alignment, if the activity is a log activity and the corresponding transition is contained in the first choice activity set, we determine whether all transitions of the whole branch headed by this transition are log activities; if they are, we regard the first and last transitions of the branch as a deviation position.

γ_1=

a	t_1	t_5	t_2	t_4	b
a	t_1	>>	t_2	t_4	b

Figure 4. An optimal alignment γ_1 between σ_1 and PN_1

γ_2=

a	t_6	t_7	c	b
a	t_6	t_7	>>	b

Figure 5. An optimal alignment γ_2 between σ_2 and PN_1

Algorithm 3 Determining deviation positions for incomplete choice structures

Input: A process tree denoted by PT, a Petri net denoted by $PN = (P, T; F, M)$, and the optimal alignment denoted by $\Gamma_{\sigma,PN,lc}$

Output: The deviation position denoted by DP_{COS}, and the new activity set denoted by AL

1: $DP_{COS} \leftarrow \phi$; $AL \leftarrow \phi$;
2: $k \leftarrow 0$; $g \leftarrow 0$; $h \leftarrow 0$;
3: $B_S \leftarrow Calculate - B_S(PT)$;
4: $[SF_P, SL_P], FC_S, LC_S \leftarrow Calculate - FLP(PT)$;
5: **for** $(i = 1; i \leq |\gamma|; i++)$ **do**
6: **if** $\pi_1(\gamma[i]) = SF_P$ **then**
7: $k \leftarrow i$;
8: **end if**
9: **if** $\pi_1(\gamma[i]) = SL_P$ **then**
10: $g \leftarrow i$;
11: **end if**
12: **if** $\pi_2(\gamma[i]) =>>$ and $\pi_1(\gamma[i]) \notin \delta(PN)$ **then**
13: $AL \leftarrow AL \cup \{(\pi_1(\gamma[i]), >>)\}$;
14: **end if**
15: **end for**
16: **for** $(i = k + 1; i < g; i++)$ **do**
17: **if** $\pi_1(\gamma[i]) \in FC_S$ and $\pi_2(\gamma[i]) =>>$ **then**
18: **for** $(j = 1; j \leq |B_S|; j++)$ **do**
19: **if** $\pi_1(\gamma[i]) \in B_S$ **then**
20: $h \leftarrow j$;
21: **for** $(v = 2; \pi_h(\pi_v(B_S)) \in \delta(\sigma); v++)$ **do**
22: Continue;
23: **end for**
24: **end if**
25: **end for**
26: **end if**
27: **end for**
28: $DP_{COS} \leftarrow DP_{COS} \cup \{(\pi_h(\pi_1(B_S)), (\pi_h(\pi_v(B_S)))\}$;
29: **return** DP_{COS}, AL

Figure 4 is an optimal alignment γ_1 between σ_1 and PN_1, and Figure 5 is an optimal alignment γ_2 between σ_2 and PN_1. σ_1 has transitions of choice structures with t_1, t_5, t_2, t_4, and the branch set of PN_1 is denoted as $B_S = \{\{t_1, t_2, t_4\}, \{t_1, t_3, t_4\}, \{t_5\}, \{t_6, t_7\}\}$. Its head-to-tail place is denoted as $[SF_P, SL_P] = [p_2, p_6]$. Its first choice activity set is denoted as $FC_S = \{t_1, t_5, t_6\}$, and the last choice activity set is denoted as $LC_S = \{t_4, t_5, t_7\}$. We can find that the branch $\{t_5\}$ occurs concurrently with the branch $\{t_1, t_2, t_4\}$ by traversing the optimal alignment. For γ_1, $(t_5, >>)$ is a log activity, and t_5 is both contained in first and last choice activity sets, so

its deviation position denoted by $DP_{COS} = \{(t_5, t_5)\}$. For γ_2, its new activity set denoted by AL is $\{c\}$.

An algorithm is proposed to repair models with incomplete choice structures via logic Petri nets according to the deviation position in the following content.

Algorithm 4 Repair models for incomplete choice structures

Input: The deviation position denoted by DP_{COS}, a Petri net denoted by $PN = (P, T; F, M)$, the head-to-tail place denoted by $[SF_P, SL_P]$, and the new activity set denoted by AL

Output: A logic Petri net denoted by $LPN' = (P', T'; F', I', O', M')$

1: $LPN' \leftarrow PN$;
2: **for** each $(t_i, t_j) \in DP_{COS}$ **do**
3: $P' \leftarrow P' \cup P_{new1} \cup P_{new2}$;
4: $T' \leftarrow T'$;
5: $F' \leftarrow F' - \{SF_P \rightarrow t_i\} - \{t_j \rightarrow SL_P\} \cup \{\{^\bullet SF_P\} \rightarrow P_{new1}\} \cup \{P_{new1} \rightarrow t_i\} \cup \{t_j \rightarrow P_{new2}\} \cup \{P_{new2} \rightarrow \{SL_P^\bullet\}\}$;
6: $I' \leftarrow I' \cup \{I(SL_P^\bullet) = \{SL_P\} \bigotimes P_{new2} \bigotimes (\{SL_P\} \wedge P_{new2})\}$;
7: $O' \leftarrow O' \cup \{O(^\bullet SF_P) = \{SF_P\} \bigotimes P_{new1} \bigotimes (\{SF_P\} \wedge P_{new1})\}$;
8: **end for**
9: **for** each $t \in AL$ **do**
10: $P' \leftarrow P' \cup P_{new}$;
11: $T' \leftarrow T' \cup t$;
12: $F' \leftarrow F' \cup \{^\triangle t \rightarrow P_{new}\} \cup \{P_{new} \rightarrow t\} \cup \{t \rightarrow SL_P\}$;
13: $O' \leftarrow O' \cup \{O(^\triangle t) = P_{new} \bigotimes \{SL_P\}\}$;
14: **end for**
15: **return** LPN

Algorithm 4 repairs models with incomplete choice structures. For each deviation position and new activity, we add different logic input and output transitions.

For σ_1 and σ_2, its deviation position and new activity set are denoted as $DP_{COS} = \{(t_5, t_5)\}$ and $AL = \{c\}$. For $DP_{COS} = \{(t_5, t_5)\}$, we delete the arc from p to t_5 and the arc from t_5 to p', where $p \in^\bullet t_5$ and $p' \in t_5^\bullet$. Then we add two places, and add the arc from a to the new place and the arc from the new place to t_5, where $a \in^\bullet p$; we also add the arc from t_5 to the new place and the arc from the new place to b, where $b \in p'^\bullet$. We change a to a logic output transition and change b to a logic input transition. For $AL = \{c\}$, $^\triangle c = t_7$ and $c^\triangle = b$. We add a place and new transition c. Then we insert three arcs from t_7 to the new place and from the new place to c and from c to b into the model. Besides, we change t_7 to a logic output transition. The repaired model by our approach denoted as LPN_2 is shown in Figure 6.

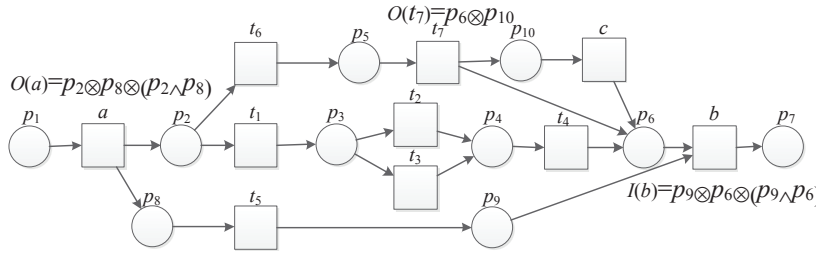


Figure 6. The repaired model LPN_2 by our approach

4 INCOMPLETE CONCURRENT STRUCTURES DEVIATION REPAIR

This section presents a repair method of the model with incomplete concurrent structures. For incomplete concurrent structures, we propose concurrent relation sets, including head-to-tail transitions and concurrent activity sets. By comparing and analyzing the logic relations between activities in traces and transitions in concurrent activity sets, we can determine the deviation position of incomplete concurrent structures, and repair models with incomplete concurrent structures via logic Petri nets.

In the following, we use $PN = (P, T; F, M)$ to denote a four-tuple Petri net, and use PT to denote a process tree of PN.

Definition 15 (Head-to-tail transition)**.** $[SF_T, SL_T]$ is called the head-to-tail transition of a concurrent structure, and SF_T and SL_T are called the head transition and the tail transition, respectively, where $SF_T = n.lsib$ and $SL_T = n.rsib$, $n =$ "$\wedge$" and $n \in PT$.

Theorem 4. For $n =$ "$\wedge$", if $\exists n.child = m$ and $m.child$=null, then $\{m\}$ is a branch.

Proof. If $\exists n.child = m$ and $m.child =$ null, it means that the child node of n is a leaf node, i.e., $\neg\exists m.child| \rightarrow \bigoplus$, where $\bigoplus = \{\times, \rightarrow, \odot, \wedge\}$. Since $n =$ "$\wedge$", $n.lsib = SF_T$ and $n.rsib = SL_T$. We have $SF_T.parent = SL_T.parent = n.parent =$ "$\rightarrow$". Since $m = n.child$ and $m.child =$ null, it means that SF_T, m and SL_T fire in order, i.e., $SF_T =^{\bullet} ({}^{\bullet}m)$ and $SL_T = (m^{\bullet})^{\bullet}$. Thus, $\{m\}$ is a branch.

If $\{m\}$ is a branch and $n =$ "$\wedge$", it means $n.lsib = SF_T$ and $n.rsib = SL_T$. We have $SF_T =^{\bullet} ({}^{\bullet}m)$ and $SL_T = (m^{\bullet})^{\bullet}$. Thus, $\neg\exists m.child| \rightarrow \bigoplus$ and "$\wedge$"$| \rightarrow m$, i.e., $m = n.child$ and $m.child =$ null. □

By Theorem 4, for the node denoted by "$\wedge$" of the process tree, if the child node of the node is a leaf node, then this child node is alone a branch in the concurrent structure.

Definition 16 (Concurrent activity set)**.** First and last concurrent activity sets are called the concurrent activity set together. FU_S and LU_S denote the first and the

last concurrent activity sets, respectively. If $n =$ "$\wedge$" and $n \in PT$, they need to satisfy:

1. if $n.child = m$, $m.child =$ null, then $FU_S = \{m\}$, and $LU_S = \{m\}$;
2. if $n| \rightarrow \ldots| \rightarrow m$, $m =$ "$\wedge$" or $m =$ "$\times$", $k = |m.child|$, $q = n.child.lp$ and $m = q.parent$, then $FU_S = \{m.child.p_1, m.child.p_2, \ldots, m.child.p_k\}$;
3. if $n| \rightarrow \ldots| \rightarrow m$, $m =$ "$\wedge$" or $m =$ "$\times$", $k = |m.child|$, $q = n.child.rp$ and $m = q.parent$, then $LU_S = \{m.child.p_1, m.child.p_2, \ldots, m.child.p_k\}$; and
4. if $n| \rightarrow \ldots| \rightarrow m$, $m =$ "$\rightarrow$", and $m.child.child =$ null, then $FU_S = \{n.child.lp\}$, and $LU_S = \{n.child.rp\}$.

The algorithm of calculating head-to-tail transitions and concurrent activity sets is given in the following.

Algorithm 5 calculates the head-to-tail transition and the concurrent activity set. For each node n of process tree, if n is "$\wedge$", its head transition is the pre-set of the pre-set of the leftmost leaf node of n, and its tail place is the post-set of the post-set of the rightmost leaf node of n. For each child node of n, if its child node is a leaf node, then the child node of n is contained in the first and last concurrent activity sets. If the child node g of n is "$\wedge$" or "$\times$", and g is the parent node of the leftmost or rightmost leaf node of n, then each child node of g is contained in the first or last concurrent activity set. If g is "$\rightarrow$" and the child node of g is a leaf node, then the leftmost leaf node of g is contained in the first concurrent activity set, and the rightmost leaf node of g is contained in the last concurrent activity set.

Figure 7 shows a Petri net model PN_2, and PN_2 contains concurrent structures. Let $\sigma_3 = \langle a, t_1, t_3, t_6, e, c\rangle$ and $\sigma_4 = \langle a, b, t_4, t_5, f, c\rangle$ be two traces. PN_2 has a concurrent structure with transitions t_1, t_2, t_3, t_4, t_5, t_6. In the trace σ_3, we have that t_1, t_3 and t_6 are concurrently enabled. For PN_2, according to Algorithm 1, we can calculate its branch set denoted by $B_S = \{\{t_1, t_3\}, \{t_2, t_3\}, \{t_4, t_5\}, \{t_6\}\}$. It shows that the branches $\{t_1, t_3\}$ and $\{t_6\}$ can occur selectively with the branch $\{t_4, t_5\}$. In the original model PN_2, all these branches need to occur. So this is an incomplete concurrent structure.

Figure 8 shows the process tree of PN_2 represented by PT_2. We have $PT_2 = \rightarrow (a, b, \wedge(\rightarrow (\times(t_1, t_2), t_3), \rightarrow (t_4, t_5), t_6), c)$. For PT_2, we can obtain the head-to-tail transition, the branch set and the concurrent activity set. Its head-to-tail transition is denoted as $[SF_T, SL_T] = [b, c]$, and its branch set is denoted as $B_S = \{\{t_1, t_3\}, \{t_2, t_3\}, \{t_4, t_5\}, \{t_6\}\}$. The first concurrent activity set is denoted as $FU_S = \{t_1, t_2, t_4, t_6\}$, and the last concurrent activity set is denoted as $LU_S = \{t_3, t_5, t_6\}$.

For incomplete concurrent structures, we can obtain log activities and model activities from the optimal alignment. For those model activities, we can obtain branches that need to occur selectively with other branches. For log activities, we can obtain new activities that need to be inserted into the original model. The algorithm of determining deviation positions is given in the following content.

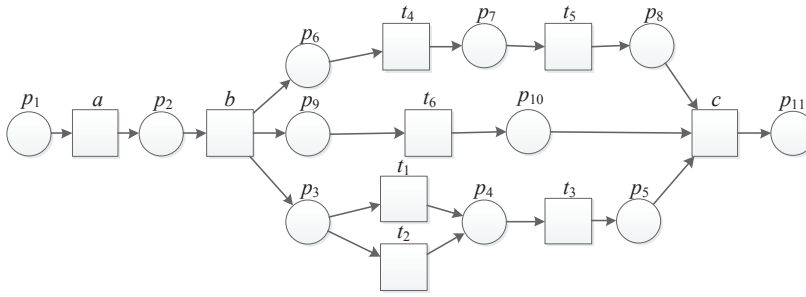


Figure 7. A Petri net model PN_2

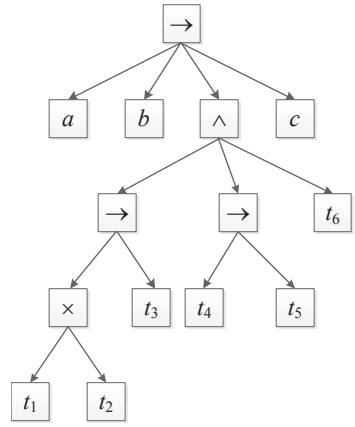


Figure 8. The process tree PT_2 of PN_2

Algorithm 6 calculates the deviation position for incomplete concurrent structures. We obtain the positions of SF_T and SL_T in the optimal alignment, and collect new activities. For activities between SF_T and SL_T in the optimal alignment, if the activity is a model activity and the corresponding transition is contained in the first concurrent activity set, we determine whether all transitions of the branch headed by this transition are model activities. If the corresponding activity of head transition is a model or synchronous activity, we regard the invisible or head transition and the first and last transitions in the branch as the deviation position.

An optimal alignment γ_3 between σ_3 and PN_2 is represented in Figure 9, and an optimal alignment γ_4 between σ_4 and PN_2 is shown in Figure 10. The branch set of PN_2 is $B_S = \{\{t_1, t_3\}, \{t_2,\ t_3\}, \{t_4, t_5\}, \{t_6\}\}$. For γ_3, $(>>, b)$, $(>>, t_4)$ and $(>>, t_5)$ are model activities, and we can find that the branch $\{t_4, t_5\}$ can occur selectively with other branches by traversing the optimal alignment. For γ_4, $(>>, t_1)$,

Algorithm 5 $Calculate - FLT(PT)$

Input: A process tree denoted by PT

Output: The head-to-tail transition denoted by $[SF_T, SL_T]$, the first concurrent activity set denoted as FU_S, and the last concurrent activity set denoted as LU_S

1: $SF_T \leftarrow \phi$; $SL_T \leftarrow \phi$; $FU_S \leftarrow \phi$; $LU_S \leftarrow \phi$; $u \leftarrow \phi$; $q_1 \leftarrow \phi$; $q_2 \leftarrow \phi$;
2: **for** each $n \in PT$ **do**
3: **if** $n =$ "$\wedge$" **then**
4: $u \leftarrow |n.child|$;
5: $m \leftarrow n.child.p_1$;
6: $k \leftarrow n.child.p_u$;
7: $SF_T \leftarrow SF_T \cup^{\bullet} ({}^{\bullet}(m.child.lp))$;
8: $SL_T \leftarrow SL_T \cup ((k.child.rp)^{\bullet})^{\bullet}$;
9: **for** $(i = 1; i \leq |n.child|; i++)$ **do**
10: $g_i \leftarrow n.child.p_i$;
11: **if** $g_i.child = null$ **then**
12: $FU_S \leftarrow FU_S \cup \{g_i.child\}$;
13: $LU_S \leftarrow LU_S \cup \{g_i.child\}$;
14: **end if**
15: $q_1 \leftarrow n.child.lp$;
16: $q_2 \leftarrow n.child.rp$;
17: **if** $g_i =$ "$\times$"$||g_i =$ "$\wedge$" and $(g_i = q_1.parent)$ **then**
18: **for** each $g_i.child$ **do**
19: $FU_S \leftarrow FU_S \cup g_i.child$;
20: **end for**
21: **end if**
22: **if** $g_i =$ "$\times$"$||g_i =$ "$\wedge$" and $(g_i = q_2.parent)$ **then**
23: **for** each $g_i.child$ **do**
24: $LU_S \leftarrow LU_S \cup g_i.child$;
25: **end for**
26: **end if**
27: **if** $g_i =$ "$\rightarrow$" and $g_i.child.child$ **then**
28: $FU_S \leftarrow FU_S \cup g_i.child.lp$;
29: $LU_S \leftarrow LU_S \cup g_i.child.rp$;
30: **end if**
31: **end for**
32: **end if**
33: **end for**
34: **return** $[SF_T, SL_T]$, FU_S, LU_S

Algorithm 6 Determining deviation positions for incomplete concurrent structures

Input: A process tree denoted by PT, a Petri net denoted by $PN = (P, T; F, M)$, and the optimal alignment denoted by $\Gamma_{\sigma,PN,lc}$

Output: The deviation position denoted by DP_{CUS}, and the new activity set denoted by AL

$DP_{CUS} \leftarrow \phi$; $AL \leftarrow \phi$;
$k \leftarrow 0$; $g \leftarrow 0$; $h \leftarrow 0$;
$B_S \leftarrow Calculate - B_S(PT)$;
$[SF_T, SL_T], FU_S, LU_S \leftarrow Calculate - FLT(PT)$;
for $(i = 1; i \leq |\gamma|; i++)$ **do**
  if $\pi_2(\gamma[i]) = SF_T$ **then**
    $k \leftarrow i$;
  end if
  if $\pi_2(\gamma[i]) = SL_T$ **then**
    $g \leftarrow i$;
  end if
  if $\pi_2(\gamma[i]) =>>$ and $\pi_1(\gamma[i]) \notin \delta(PN)$ **then**
    $AL \leftarrow AL \cup \{(\pi_1(\gamma[i]), >>)\}$;
  end if
end for
for $(i = k + 1; i < g; i++)$ **do**
  if $\pi_2(\gamma[i]) \in FU_S$ and $\pi_1(\gamma[i]) =>>$ **then**
    for $(j = 1; j \leq |B_S|; j++)$ **do**
      if $\pi_2(\gamma[i]) \in B_S$ **then**
        $h \leftarrow j$;
        for $(v = 2; \pi_h(\pi_v(B_S)) \in \delta(\sigma); v++)$ **do**
          Continue;
        end for
      end if
    end for
  end if
end for
if $\pi_1(\gamma[k]) =>>$ **then**
  $DP_{CUS} \leftarrow DP_{CUS} \cup \{\tau(\pi_h(\pi_1(B_S)), (\pi_h(\pi_v(B_S)))\}$;
else
  $DP_{CUS} \leftarrow DP_{CUS} \cup \{\pi_1(\gamma[k])(\pi_h(\pi_1(B_S)), (\pi_h(\pi_v(B_S)))\}$;
end if
return DP_{CUS}, AL

$(>>, t_3)$ and $(>>, t_6)$ are model activities, and we can find branches $\{t_1, t_3\}$ and $\{t_6\}$ occur selectively with other branches. Its head-to-tail transition is $[SF_T, SL_T] = [b, c]$, its first concurrent transition is $FU_S = \{t_1, t_2, t_4, t_6\}$, and the last concurrent transition is $LU_S = \{t_3, t_5, t_6\}$. Because $\{t_4, t_5\}$ is a branch of PN_2, and the head transition b also cannot fire, so $\{\tau(t_4, t_5)\}$ is a deviation. Because $\{t_1, t_3\}$ and $\{t_6\}$ are two branches of PN_2, and the head transition b is a synchronous activity, so $\{b(t_1, t_3), b(t_6, t_6)\}$ is a deviation. So we obtain its deviation position is $DP_{CUS} = \{\tau(t_4, t_5)), b(t_1, t_3), b(t_6, t_6)\}$. Besides, $(e, >>)$ and $(f, >>)$ are log activities, the new transitions e and f need to be inserted into the model, and its new activity set is $AL = \{e, f\}$.

γ_3=

a	>>	t_1	t_3	t_6	>>	>>	e	c
a	b	t_1	t_3	t_6	t_4	t_5	>>	c

Figure 9. An optimal alignment γ_3 between σ_3 and PN_2

γ_4=

a	b	t_4	t_5	>>	>>	>>	f	c
a	b	t_4	t_5	t_1	t_3	t_6	>>	c

Figure 10. An optimal alignment γ_4 between σ_4 and PN_2

Algorithm 7 repairs models for incomplete concurrent structures via logic Petri nets according to the deviation position. For different deviation positions, we add different logic input and output transitions.

For σ_3, σ_4 and PN_2, its deviation position is denoted by DP_{CUS}=$\{\tau(t_4, t_5))$, $b(t_1, t_3)$, $b(t_6, t_6)\}$, and its new activity set is denoted by $AL = \{e, f\}$. For $\tau(t_4, t_5))$, we add an invisible transition to skip transition b, and add two arcs from the invisible transition to p and p', where $p \in^\bullet t_1$ and $p' \in^\bullet t_6$. For $b(t_1, t_3)$ and $b(t_6, t_6)$, we change the head transition b to a logic output transition. For $AL = \{e, f\}$, we add two places and new transitions e and f, and change the tail transition c to a logic input transition. The model repaired by our approach denoted by LPN_3 is represented in Figure 11.

5 EXPERIMENTAL EVALUATION

This section will compare our repair approach with Fahland's approach, Knapsack's approach and Goldratt's approach. The data is from a routine examination and a CT index examination of a hospital in Qingdao, and event logs can be accessible at: `https://pan.baidu.com/s/1Nx2vf82NYKB9TGbf8uYIFQ`. The Fahland's approach is implemented in ProM6.6, which is a process mining tool with lots of plugins and can be available from: `http://www.promtools.org/prom6/`. The Goldratt's approach and Knapsack's approach are implemented in the DOS window and edited

Algorithm 7 Repair models for incomplete concurrent structures

Input: The deviation position denoted by DP_{CUS}, a Petri net denoted by $PN = (P, T; F, M)$, the head-to-tail transition denoted by $[SF_T, SL_T]$, and the new activity set denoted by AL

Output: A logic Petri net denoted by $LPN' = (P', T'; F', I', O', M')$

1: $LPN' \leftarrow PN$;
2: **for** each $SF_T(t_i, t_j) \in DP_{CUS}$ **do**
3: $P' \leftarrow P'$;
4: $T' \leftarrow T'$;
5: $F' \leftarrow F'$;
6: $O' \leftarrow O' \cup \{O(SF_T = [\wedge\{^\bullet t_i\}] \bigotimes \{\{SF_T^\bullet\} - \{^\bullet t_i\}\}\}$;
7: $I' \leftarrow I' \cup \{I(SL_T) = [\wedge\{t_j^\bullet\}]\}$;
8: **end for**
9: **for** each $\tau(t_i, t_j) \in DP_{CUS}$ **do**
10: $P' \leftarrow P'$;
11: $T' \leftarrow T' \cup \{\tau\}$;
12: $F' \leftarrow F' \cup \{\{^\bullet SF_T\} \rightarrow \tau\} \cup \{\tau \rightarrow \{\{SF_T^\bullet\} - \{^\bullet t_i\}\}\}$;
13: **end for**
14: **for** each $t \in AL$ **do**
15: $P' \leftarrow P' \cup P_{new}$;
16: $T' \leftarrow T' \cup t$;
17: $F' \leftarrow F' \cup \{\{(^\triangle t)^\bullet\} \rightarrow t\} \cup \{t \rightarrow P_{new}\} \cup \{P_{new} \rightarrow SL_T\}$;
18: $I' \leftarrow I' \cup \{I(SL_T) = I(SL_T) \cup [\bigotimes P_{new}]\}$;
19: **end for**
20: **return** LPN

in ProM6.6. Since there are no corresponding experimental tools for mining and repairing logic Petri nets, the model repair and analysis of our repair approach use manual simulation in this paper.

5.1 Experiment Data

We take two business processes from a routine examination and a CT index examination in a hospital as examples. A hospital routine examination business process is represented in Figure 12. First, a patient makes an appointment in the hospital and pays for an appointment. Then he (or she) can get a number and wait for his (or her) order. After that, a doctor will check what the patient needs by the routine examination. There are five types of examinations and the patient can do one of them, i.e., the electrocardiogram, the abdominal ultrasound, the lung function examination, the blood glucose and lipid, and the liver and kidney examination. Finally, a doctor will diagnose and cure disease according to examinations. Figure 13 shows a hospital CT index examination business process. First, a patient goes to the information desk to consult some related problems and makes an appointment.

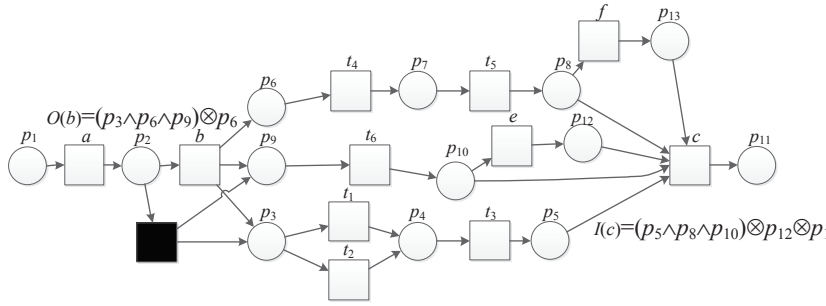


Figure 11. The repaired model LPN_3 by our approach

Then he (or she) will have an outpatient examination and take a brain CT and a head CT. Besides, a doctor checks for four symptoms, i.e., the sinusitis, the brain damage, the cerebral infarction, and the intracranial tumor. After that, the patient needs a surgery and is hospitalized for tests.

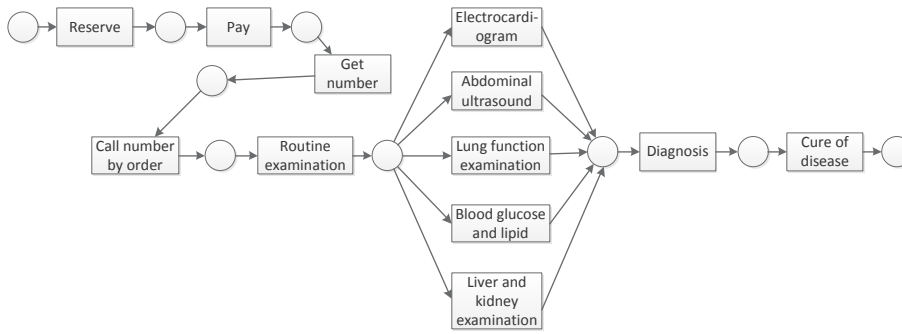


Figure 12. A hospital routine examination business process

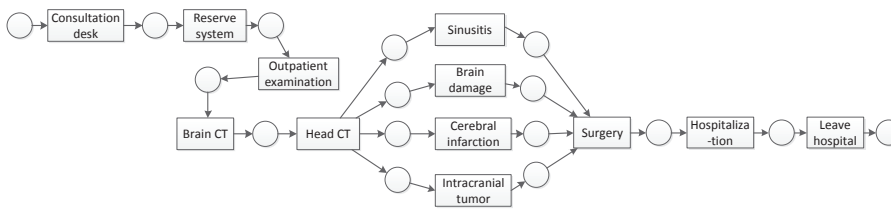


Figure 13. A hospital CT index examination business process

However, some event logs deviating from the original model are generated in the real process model systems. For example, in the hospital routine examination

business process, a patient can do four tests at once, i.e., the liver and kidney examination, the lung function examination, the blood glucose and lipid, and the electrocardiogram; the patient also can carry out the abdominal ultrasound and the gynecological examination together. In the hospital CT index examination business process, the patient may only have a brain examination, he (or she) is tested for only three conditions, i.e., the intracranial tumor, the brain damage, and the cerebral infarction; the patient also can take medicines after the sinusitis examination. In those situations, the process models need to be repaired, the choice structure needs to be repaired to an incomplete choice structure, and the concurrent structure needs to be repaired to an incomplete concurrent structure. These two repaired structures can describe both choice and concurrent structures. Petri net-based models cannot simply and accurately express those structures, and we can repair model based on logic Petri nets.

5.2 Model Repair

For event logs of a hospital routine examination business process, we first filter out event logs that are significantly deviated from the examination business process. And according to the preprocessed ten sets of event logs (as shown in Table 1), the process model (as shown in Figure 12) is repaired based on four model repair methods. Table 1 records the specific information of activities in event logs and the number of deviations.

Logs	Traces	Events	Transitions	Length	Deviations
L_1	100	1 020	13	8–11	220
L_2	200	2 103	13	8–12	503
L_3	300	3 143	13	8–12	743
L_4	400	4 306	13	8–12	1 106
L_5	500	5 447	13	8–12	1 447
L_6	600	6 129	13	8–11	1 378
L_7	700	7 169	13	8–11	1 570
L_8	800	8 179	13	8–11	1 780
L_9	900	9 169	13	8–11	1 970
L_{10}	10 00	10167	13	8–11	2 168

Table 1. Event logs L_1–L_{10} of a routine examination business process

Our proposed approach is compared with Fahland's approach, Knapsack's approach and Goldratt's approach to illustrate its correctness and effectiveness. For incomplete choice structures, the Fahland's approach repairs the choice structure by adding loop structures, and adds invisible transitions to skip transitions that are not enabled. The Goldratt's method and Knapsack's method add different self-loops of repeat transitions to improve the fitness of process models. For incomplete concurrent structures, the Fahland's approach collects new transitions as a sub-log and inserts it into the original model, and skips transitions that are not

enabled by adding invisible transitions. Besides, the Goldratt's method and Knapsack's method also add many invisible transitions to make repaired models better replay event logs generated in real systems. These invisible transitions and repeat transitions increase the uncertainty of the operation of the model, which leads to a poor performance of the model. These models cannot represent the logic relation among activities in incomplete choice structures and incomplete concurrent structures well, and most of them deviate from original structures of process models. It does not take advantage of the application and extension of process models.

For Figure 12 and Table 1, models repaired by four approaches are represented in Figures 14, 15, 16 and 17, respectively. The model repaired by Fahland's method adds a loop structure to make transitions in the choice structure fire simultaneously. The models repaired by Goldratt's approach and Knapsack's approach add different self-loops of single transitions. Our approach does not add any repeat transitions and does not add loop structures to change the basic structure of the original model. The model repaired by our method contains 3 logic transitions, the logic input function is $I(Diagnosis) = p_{10} \bigotimes p_{11} \bigotimes p_{13} \bigotimes p_{14} \bigotimes (p_{10} \wedge p_{11} \wedge p_{13} \wedge p_{14})$, and the logic output functions are $O(Routine\ examination) = p_6 \bigotimes p_7 \bigotimes p_8 \bigotimes p_9 \bigotimes (p_6 \wedge p_7 \wedge p_8 \wedge p_9)$ and $O(Abdominal\ ultrasound) = p_{12} \bigotimes p_{13}$. Our repaired model can describe the logic relation among transitions of incomplete choice structures. These input and output functions limit the enablement of transitions in the model, so the model generates fewer traces that are not included in event logs.

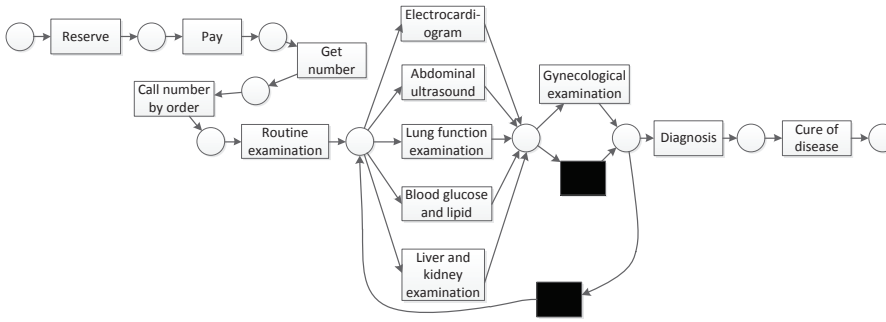


Figure 14. The repaired routine examination model by Fahland's method

Compared with the original model, the addition results of four repair models are represented in Table 2. As shown in Table 2, Goldratt's method adds the largest number of transitions (including invisible transitions), and our method only adds one transition without invisible transitions. The number of added repeat transitions by Goldratt's method and Knapsack's method are 6 and 4, respectively. Compared with other three repair methods, our method adds the least transitions, and does not add invisible transitions and repeat transitions.

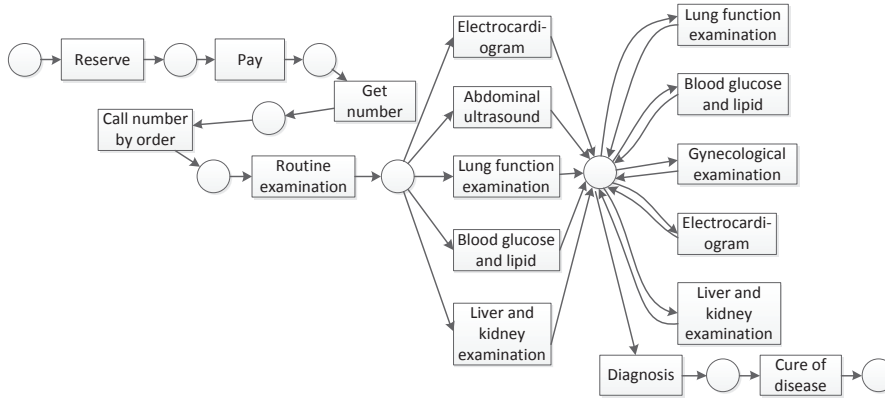


Figure 15. The repaired routine examination model by Knapsack's method

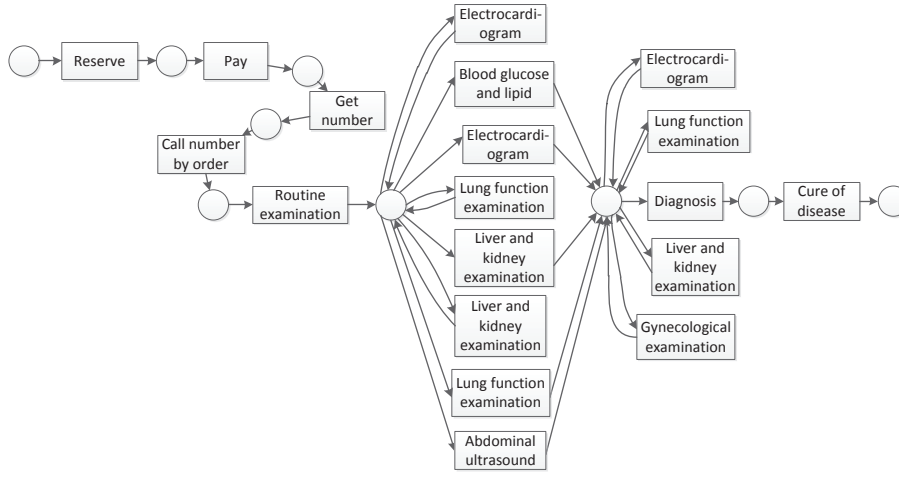


Figure 16. The repaired routine examination model by Goldratt's method

Four Repair Methods	Added $\|P\|$	Added $\|T+\tau\|$	Added $\|F\|$	Added Repeat $\|T\|$
Our method	7	1	9	0
Fahland's method	1	3	6	0
Goldratt's method	0	7	14	6
Knapsack's method	0	5	10	4

Table 2. The addition results of Figures 14, 15, 16 and 17

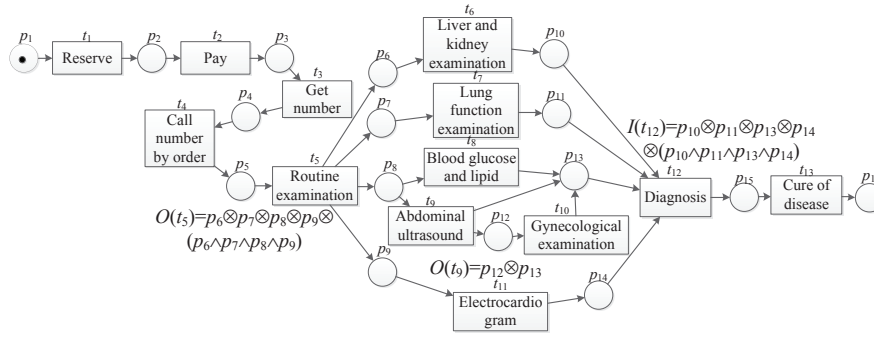


Figure 17. The repaired routine examination model by our approach

We process event logs of a CT index examination business process according to the same data filtering method. The filtered ten sets of event logs are described in Table 3. The process model (as shown in Figure 13) is repaired based on ten sets of event logs. The information of activities in event logs and the number of deviations are recorded in Table 3.

Logs	Traces	Events	Transitions	Length	Deviations
L_{11}	100	1 166	13	10–12	38
L_{12}	200	2 366	13	9–12	38
L_{13}	300	3 366	13	9–12	238
L_{14}	400	4 566	13	9–12	238
L_{15}	500	5 765	13	9–12	238
L_{16}	600	6 965	13	9–12	238
L_{17}	700	8 165	13	9–12	238
L_{18}	800	9 100	13	9–12	573
L_{19}	900	10 300	13	9–12	573
L_{20}	1 000	11 500	13	9–12	573

Table 3. Event logs $L_{11} - L_{20}$ of a CT index examination business process

Four repaired models repaired by three classic methods and our repair method are shown in Figure 18, 19, 20 and 21, respectively. Fahland's method inserts many invisible transitions into the model to skip concurrent transitions that cannot fire. Although it makes activities in event logs be well replayed in the model, it also reduces the precision and simplicity of the model. Goldratt's approach inserts activities deviating from the model into the original model by means of self-loops. Knapsack's approach repairs the model in the same way as Goldratt's approach with different constraints. Besides, these two methods add a lot of invisible transitions. The model repaired by our approach reduces the degree of uncertainty of transition firing and has a high simplicity of net structures. Besides, it contains one logic input function and one logic output function, and they are

$I(Surgery) = (p_{10} \wedge p_{11} \wedge p_{12} \wedge p_{13}) \bigotimes p_{14} \bigotimes (p_{10} \wedge p_{11} \wedge p_{12})$ and $O(Head\ CT) = (p_6 \wedge p_7 \wedge p_8 \wedge p_9) \bigotimes p_9$. The repaired model by our approach can describe the logic relation among transitions of incomplete concurrent structures, and it does not generate additional transitions to increase the uncertainty of the operation of the model.

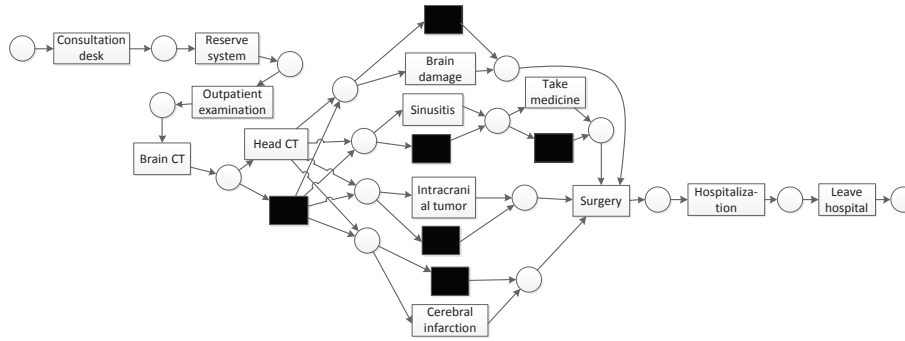


Figure 18. The repaired CT index examination model by Fahland's method

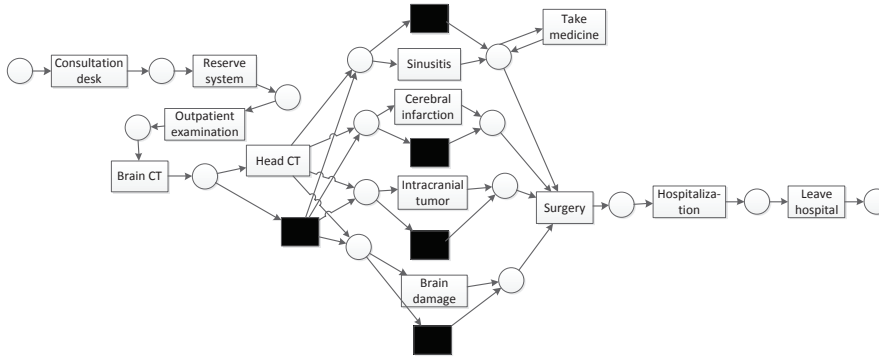


Figure 19. The repaired CT index examination model by Knapsack's method

Table 4 records addition results of four repair methods. Fahland's method adds the maximum number of transitions, including invisible transitions. The addition result of our repaired model is best, and the other three repaired models add a lot of invisible transitions and repeat transitions.

5.3 Performance Analysis

The sub-section describes the performance analysis of four repaired models combining with event logs L_1–L_{20}. Fitness is the proportion of traces that the model

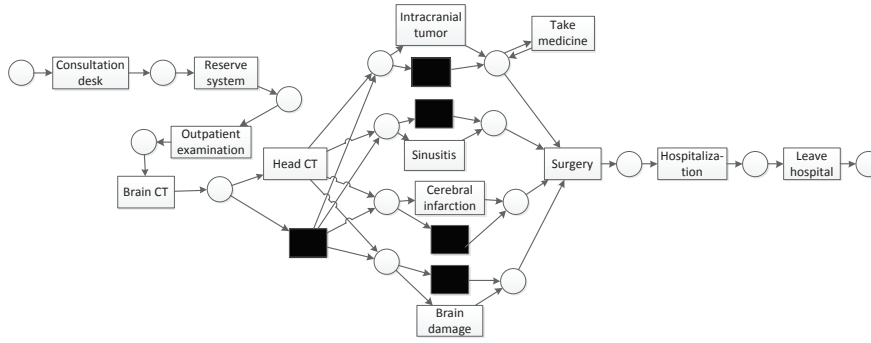


Figure 20. The repaired CT index examination model by Goldratt's method

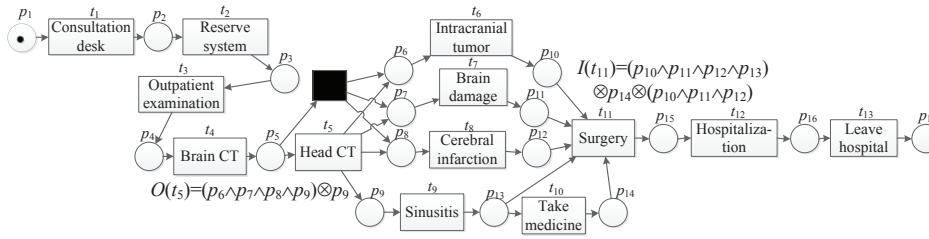


Figure 21. The repaired CT index examination model by our approach

can completely replay in the event log. A model with high fitness allows behaviors described in the event log to occur. If a model is precise, it does not allow too many activities that are not described in event logs to appear in the model. The formula for calculating fitness and precision of logic Petri nets are proposed in [19]. We consider the structure of the model and the repeatability of activities, and calculate the simplicity of net-based structures according to the method proposed in [18].

For four repaired models, we obtain the degree of fitness between models and event logs L_1–L_{20} represented in Table 5 and Table 6. As shown in Table 5, for our approach, Fahland's approach and Goldratt's approach, the fitness be-

Four Repair Methods	Added $\|P\|$	Added $\|T+\tau\|$	Added $\|F\|$	Added Repeat $\|T\|$
Our method	1	2	7	0
Fahland's method	1	7	17	0
Goldratt's method	0	6	15	1
Knapsack's method	0	6	15	1

Table 4. The addition results of Figures 18, 19, 20 and 21

tween these repaired models and event logs L_1–L_{10} are all 1. However, the fitness value of the model repaired by Goldratt's method is slightly lower than that of the other methods. As shown in Table 6, the degree of fitness between these four repaired models and event logs L_{11}–L_{20} are all 1. In general, the fitness of these four methods is very high, and those four repaired models all have a high fitness.

Four Repair Methods	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
Fahland's method	1	1	1	1	1	1	1	1	1	1
Our method	1	1	1	1	1	1	1	1	1	1
Goldratt's method	1	1	1	1	1	1	1	1	1	1
Knapsack's method	1	0.9884	0.9884	0.9723	0.9710	1	1	1	1	1

Table 5. The fitness between different models and event logs L_1–L_{10}

Four Repair Methods	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
Fahland's method	1	1	1	1	1	1	1	1	1	1
Our method	1	1	1	1	1	1	1	1	1	1
Goldratt's method	1	1	1	1	1	1	1	1	1	1
Knapsack's method	1	1	1	1	1	1	1	1	1	1

Table 6. The fitness between different models and event logs L_{11}–L_{20}

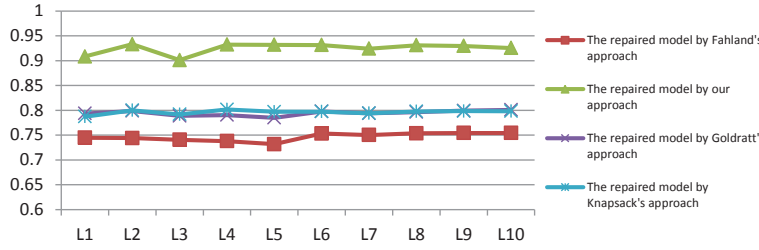


Figure 22. The precision between different models and event logs L_1–L_{10}

The results of precision between models proposed by four repair methods and event logs L_1–L_{20} are represented in Figure 22 and Figure 23. As shown in Figure 22, the model repaired by our approach has the highest precision, significantly higher than Knapsack's approach, Goldratt's approach and Fahland's approach. As shown in Figure 23, the model repaired by our approach also has the highest precision, and it has a clear performance advantage than the other three repair methods. In general, the precision of our repaired model for incomplete choice structures or incomplete concurrent structures is significantly higher than that of the three other classic repair methods. Our proposed method greatly improves the

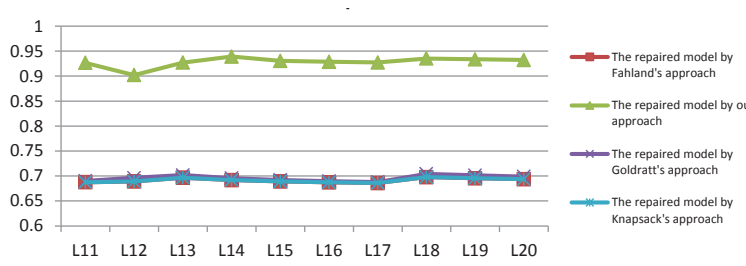


Figure 23. The precision between different models and event logs L_{11}–L_{20}

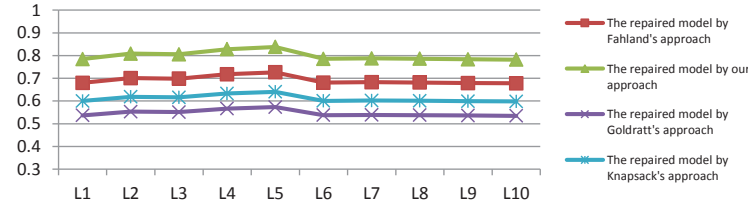


Figure 24. The simplicity between different models and event logs L_1–L_{10}

precision of the process models and also improves the performance of the models.

The simplicity of net structures is obtained by calculating the sum of the proportion of events of per trace in the total number of transitions of models. The results of simplicity of four repair approach combining with event logs L_1–L_{20} are represented in Figure 24 and Figure 25. As shown in Figure 24, the simplicity of the repaired model by our approach is highest, followed by Fahland's approach. These two methods are higher than Knapsack's approach and Goldratt's approach. As shown in Figure 25, our repair approach has the highest simplicity of net structures, higher than Goldratt's approach and Knapsack's approach. By comparison, the repaired model by Fahland's approach has the lowest simplicity of net structures. In terms of net structures, our repair approach has the highest simplicity of net structures, higher than Goldratt's approach, Knapsack's approach

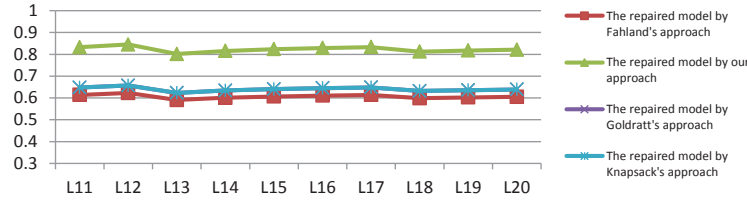


Figure 25. The simplicity between different models and event logs L_{11}–L_{20}

and Fahland's approach. Therefore, for business processes with incomplete concurrent and choice structures, our method not only enables the model to have a high precision and fitness, but also improves the simplicity of net structures of the model.

6 CONCLUSIONS

In the paper, the repair method for process models with incomplete choice and concurrent structures is proposed via logic Petri nets. Current model repair methods tend to change the choice structure to a loop structure, and generate a large number of invisible and repeat transitions, or add self-loops of transitions to improve fitness. The concepts of choice relation sets and concurrent relation sets are presented based on process trees. For incomplete choice structures, we judge whether the corresponding transitions of log activities are contained in a whole branch, and determine deviation positions based on choice activity sets. For incomplete concurrent structures, we find deviations by judging whether the corresponding transitions of model activities are contained in a whole branch and whether the head-to-tail transition is a synchronous activity. Then we determine deviation positions based on concurrent activity sets, and repair models via logic Petri nets according to different deviation positions. Through the simulation experiment, we prove that our repair approach greatly improves the precision and simplicity of the model, and the fitness is still very high. It also can describe the logic relation among transitions in incomplete choice and concurrent structures correctly. However, process trees can only represent Petri nets with block-distributed structures. For those process models that cannot be represented by process trees, we will further study how to determine the deviation position through other algorithms.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant 61903229 and Grant 61973180, 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 and Grant ZR2019BF004.

REFERENCES

[1] Qi, H. D.—Du, Y. Y.—Liu, W.: Process Model Repairing Method Based on Reachable Markings. Journal of Shandong University of Science and Technology (Natural Science), Vol. 36, 2017, No. 1, pp. 118–124.

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

[3] van der Aalst, W. M. P.—ter Hofstede, A. H. M.: YAWL: Yet Another Workflow Language. Information Systems, Vol. 30, 2004, No. 4, pp. 245–275, doi: 10.1016/j.is.2004.02.002.

[4] Liu, G. J.: Complexity of the Deadlock Problem for Petri Nets Modeling Resource Allocation Systems. Information Sciences: An International Journal, Vol. 363, 2016, Iss. C, pp. 190–197, doi: 10.1016/j.ins.2015.11.025.

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

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

[7] Weijters, A. J. M. M.—van der Aalst, W. M. P.: Rediscovering Workflow Models from Event-Based Data Using Little Thumb. Integrated Computer-Aided Engineering, Vol. 10, 2003, No. 2, pp. 151–162.

[8] Adriansyah, A.—Munoz-Gama, J.—Carmona, J.—van Dongen, B. F.—van der Aalst, W. M. P.: Aligning Based Precision Checking. In: La Rosa, M., Soffer, P. (Eds.): Business Process Management Workshops (BPM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 132, pp. 137–149, doi: 10.1007/978-3-642-36285-9_15.

[9] Leemans, S. J. J.—Fahland, D.—van der Aalst, W. M. P.: Scalable Process Discovery and Conformance Checking. Software and Systems Modeling, Vol. 17, 2018, pp. 599–631, doi: 10.1007/s10270-016-0545-x.

[10] Alizadeh, M.—Lu, X.—Fahland, D.—Yanonne, N.—van der Aalst, W. M. P.: Linking Data and Process Perspectives for Conformance Analysis. Computers and Security, Vol. 73, 2018, pp. 172–193, doi: 10.1016/j.cose.2017.10.010.

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

[12] Buijs, J. C. A. M.—van Dongen, B. F.—van der Aalst, W. M. P.: A Genetic Algorithm for Discovering Process Trees. IEEE Congress on Evolutionary Computation, 2012, 8 pp., doi: 10.1109/CEC.2012.6256458.

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

[14] Polyvyanyy, A.—van der Aalst, W. M. P.—ter Hofstede, A. H. M.—Wynn, M. T.: Impact-Driven Process Model Repair. ACM Transactions on Software Engineering and Methodology, Vol. 25, 2016, No. 4, pp. 25–33, doi: 10.1145/2980764.

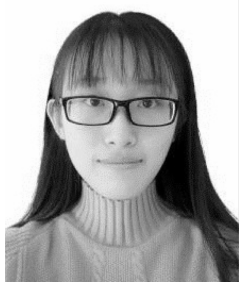
[15] Qi, H. D.—Du, Y. Y.—Qi, L.—Wang, L.: An Approach to Repair Petri Net-Based Process Models with Choice Structures. Enterprise Information Systems, Vol. 12, 2018, No. 8-9, pp. 1149–1179, doi: 10.1080/17517575.2018.1432768.

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

[17] Zhang, X. Z.—Du, Y. Y.—Qi, L.—Sun, H. C.: An Approach for Repairing Process Models Based on Logic Petri Nets. IEEE Access, Vol. 6, 2018, pp. 29926–29939, doi: 10.1109/ACCESS.2018.2843137.

[18] Teng, Y. X.—Du, Y. Y.—Qi, L.—Luan, W. J.: A Logic Petri Net-Based Method for Repairing Process Models with Concurrent Blocks. IEEE Access, Vol. 7, 2019, pp. 8266–8282, doi: 10.1109/ACCESS.2018.2890070.

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



Yuanxiu Teng received her B.Sc. degree from Shandong University of Science and Technology, Qingdao, China, in 2017. She is now pursuing the M.Sc. degree in the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China. Her current research interests are process mining, Petri nets and workflow.

Liang Qi received his B.Sc. degree in information and computer science and M.Sc. degree in computer software and theory from 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. He is currently Lecturer of computer science and technology at Shandong University of Science and Technology, Qingdao, China. His current research interests include Petri nets, discrete event systems, process mining and optimization algorithms.

Yuyue Du received his B.Sc. degree from Shandong University, Jinan, China, in 1982, the M.Sc. degree from 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 Professor at the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China. His research interests are in formal engineering, Petri nets, real-time systems, process mining and workflows.