IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 4, 2019, accepted June 13, 2019, date of publication June 18, 2019, date of current version July 3, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2923624

A Method for Mining Process Models With Indirect Dependencies via Petri Nets

HUIMING SUN, YUYUE DU, LIANG QI, (Member, IEEE), AND ZHAOYANG HE
College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China

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

This work was supported in part by the Taishan Scholar Construction Project of Shandong Province, in part by the Key Research and Development Program of Shandong Province under Grant 2018GGX101011, Grant ZR2019BF004, and Grant ZR2019BF041, in part by the Natural Science Foundation of Shandong Province under Grant ZR2018MF001, and in part by the Shandong University of Science and Technology under Grant SDKDYC190224.

ABSTRACT Process mining aims to build the models of business processes and get valuable information according to event logs generated from enterprise information systems. There exist some indirect dependences, which refer to the relationship between discontinuous activities in business processes. However, the existing approaches cannot accurately identify such dependences from the event logs. Thus, this paper extends the α algorithm and proposes a new one named the α^{TR} algorithm, which uses the association rules to describe the indirect dependences. First, an algorithm is proposed to identify the choice and loop structures in the business process. Then, the association rules are mined to describe the indirect dependences. Finally, we design an extended Petri net to formalize the process model, which can accurately describe the indirect dependences. The effectiveness of the proposed approach is illustrated by the experiments on ProM.

INDEX TERMS Process mining, process model, indirect dependency, association rule, Petri net.

I. INTRODUCTION

Nowadays, most enterprises are using information systems to manage complex business processes. For example, resource planning systems and customer relationship management systems have significantly improved the efficiency of enterprise operations. At the same time, a large number of event logs are generated. Discovering valuable information from the event logs becomes a hot research topic. Process mining is an emerging technique that aims to mine valuable process-related information from event logs and help improve the efficiency of the business process [1]. It usually has the following four mining perspectives: (1) control-flow perspective, (2) organizational perspective, (3) case perspective, and (4) time perspective. Control-flow perspective focuses on the occurrence sequence of activities. Organizational perspective studies the resources in the event logs. Case perspective mainly considers the properties of activities. Time perspective is concerned with the occurrence time and frequency of activities, which can help us discover some process bottlenecks and improve service efficiency. Process mining mainly has the following three applications: (1) process discovery, (2) conformance checking, and (3) process enhancement. Discovery algorithms are used to generate process models from event logs. Process models can reflect the business processes of the enterprise. It can be regarded as the link between the real business process and the event logs. The process model is usually constructed via a Petri net, c-net, and business process modeling notation (BPMN).

There are four dimensions to measure the quality of a mined process model: (1) fitness, (2) simplicity, (3) precision, and (4) generalization [1]. Fitness represents the capability of a process model to replay a sequence of activities in event logs. Simplicity representing the simplification of process models requires the least number of nodes and simplest structures. Precision refers to whether the activities of the process model are consistent with those in event logs. Generalization indicates that the model can allow more behaviors than those in the event logs which may appear in the future event logs. The aforementioned four dimensions are used to evaluate the discovered process model.

Process discovery has many challenges such as dealing with duplicate activities [2], non-free-choice structures [2], short loop structures [3], noise [1], and incompleteness [1]. Many scholars propose several algorithms. α algorithm proposed in [3] is one of the most classical process mining

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

algorithms. It uses the dependency of activities to generate a process model. It can identify a parallel structure in the model, but cannot mine short loop structures and non-free-choice structures. Many algorithms are thus proposed by extending α algorithm. α^+ algorithm proposed in [4] can correctly mine process models with short loop structures by analyzing the characteristics of the structures. α^{++} algorithm can build a model with non-free-choice structures [5]. $\alpha^{\#}$ algorithm proposed in [6] is used to mine invisible activities. Li *et al*. propose α^* algorithm to correctly mine a model containing duplicate activities from event logs [7]. In addition, intelligent domain technologies have also been well applied in process mining. For example, Medeiros *et al*. use the idea of a genetic algorithm for process mining [8]. This method can handle noise and incomplete event logs. However, it is not efficient for dealing with large-scale event logs. In [9] and [10], van der Aalst *et al*. and Cortadella *et al*. use the idea of state-based regions, which can express more complex control-flow structures and better balance the ''over-fitting'' and ''under-fitting'' problem. The models obtained by the methods can ensure good fitness. However, these methods will lead to state space explosions when mining large-scale event logs. [11]–[13] use the idea of language-based regions by adding nodes to models. These methods can effectively mine loop structures, but it does not allow invisible activities in the event logs. A fuzzy mining algorithm is proposed in [14], which has advantages in dealing with noise and incomplete event logs. The model obtained by these algorithms is relatively simple. In [15] and [16], approaches are proposed to describe a Heuristics Miner (HM) via c-net. They focus on the frequency and sequence of activities in the event logs, therefore, having a good capability to handle noise in event logs. Inductive Miner (IM) proposed in [17] uses the idea of linear programming to mine block structures. It solves the problem of noise and can obtain a model with high fitness. An algorithm proposed in [18] uses the distance of the traces, which can deal with the noise well. Besides, it proposes decision rules of parallel, loop, and choice structures.

The dependencies of activities are the basis of many mining algorithms. These dependencies are usually divided into direct dependencies and indirect dependencies. Direct dependency refers to direct causal relationships of continuous activities, which is also called explicit dependency in [5]. Indirect dependency reflecting the indirect causal relationships of non-continuous activities is also called implicit dependency [5]. Figure 1 shows three parts that contain many activities in a process. Since A and B are continuous, there are direct dependencies between A and B. Since A and C are discontinuous, there are indirect dependencies between A and C. From the case perspective, it is easy to mine the aforementioned two dependencies according to the properties of activities. For example, Kalynychenko *et al*. [19] consider the time dimension of activities and can correctly mine indirect dependencies. Sarno *et al.* [20] mine the decision points in process models. In addition, some methods based on decision trees have a good performance in finding

FIGURE 1. Three parts in a process.

dependencies. However, from the control-flow perspective, the existing algorithms can only mine direct dependencies and mining indirect dependencies is still a big challenge.

As shown in Figure 2, a process of password verification exists in most of the information systems. In order to protect one's account security, the systems will freeze an account after entering a wrong password multiple times. Therefore, there is an indirect dependency between account frozen and the number of times that a wrong password is reentered. Freezing an account is regarded as an activity in a choice structure. Reentering and verifying a password and verification failed are activities in a loop structure. Thus, we call the choice structure affected by the loop count as a loop-count-driven-choice structure. The existing algorithms cannot mine this kind of structure. Besides, indirect dependencies vary for different systems. Their mining is important for building a model to describe and optimize the system.

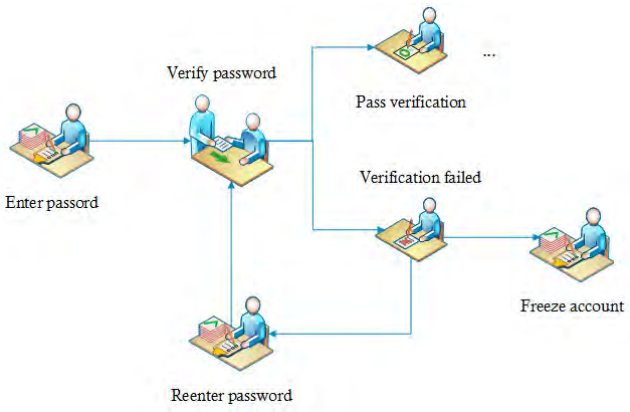


FIGURE 2. A model of password verification.

This work adopts a four-layer process mining framework as shown in Figure 3. The bottom layer represents real process systems generating event logs. The next layer indicates that the event logs are stored in the database as eXtensible event stream files. Then as shown in the second layer, we design some algorithms to identify indirect dependencies and obtain process models. We use association rules [1] to describe indirect dependencies in the models. Finally, as shown in the top layer, since the implicit dependencies and models are already obtained, they could help us find problems in the real process and optimize the process in the future. This work will propose algorithms to mine event logs and obtain process models and indirect dependencies. Our algorithm can better reflect the real process than these algorithms that only mine

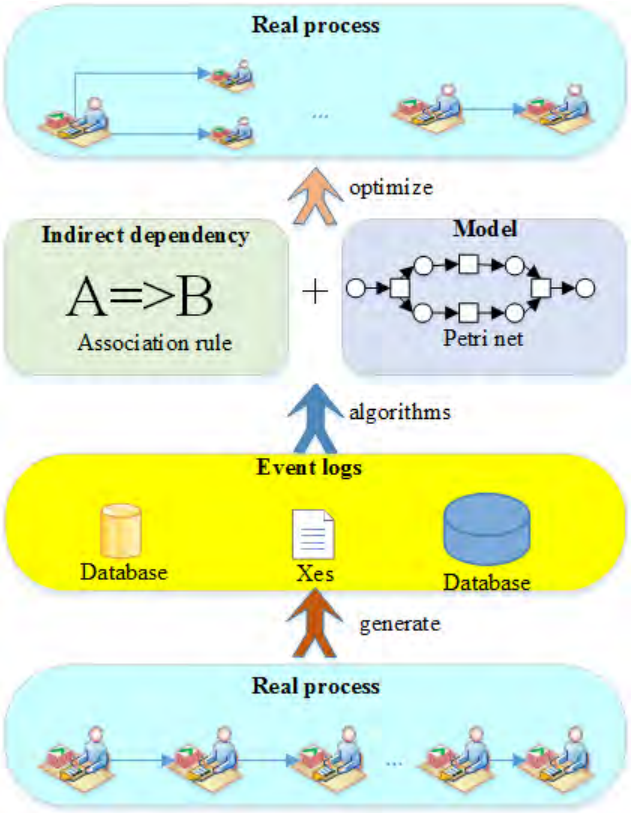


FIGURE 3. A four-layer process mining framework.

models. The contributions of the paper are summarized as follows:

(1) A method to identify choice and loop structures is proposed.
(2) A method to mine association rules is presented.
(3) In order to solve the problem that the Petri net and existing extended Petri nets cannot express indirect dependencies, a new extend Petri net containing association rules is designed.
(4) A new algorithm for mining process models with indirect dependencies is proposed.

The rest of the paper is organized as follows. Section II reviews some basic concepts of Petri nets and α algorithm. In Section III, an algorithm of mining models with indirect dependencies is proposed. Section IV shows the experiments and analyzes the results. In Section V, we conclude this paper and discuss future work.

II. PRELIMINARIES

This section first reviews the concepts and notions of sequence [21]–[24], multi-set [25], [26], trace [27], [28], event log [29]–[31], Petri net [32]–[43], and indirect dependency [5]. Petri net can be used not only for modeling [41]–[43], but also for cycle-time analysis [32], resource management [33]-[35], detecting errors [36], and service net analysis [39]. Then the ordering relations [5] and α algorithm are introduced.

Definition 1 (Sequence): Let ψ be a set. $s =< s[0], s[1], \ldots, s[n-1] >$ is a sequence over ψ, where $s[i]$ denotes the i-th element of s, and $|s| = n$ denotes the length of s. ψ^* is the set of all sequences over ψ. ε denotes an empty sequence.

Definition 2 (Multi-Set): Let ψ be a set. A multi-set D over ψ is a function D: $\psi \rightarrow N_+$, where N_+ represents a set of positive integers. $\boldsymbol{B}(\psi)$ denotes the set of all multi-sets over ψ.

Definition 3 (Trace): Let ψ be a set of activities. A trace $\sigma \in \psi^*$ is a sequence of activities on ψ, where $|\sigma| \geq 2$.

Definition 4: Let ψ be a set of activities. For $\forall\sigma \in \psi^*$, $\partial_{\text{set}}(\sigma)$ is a set of activities in σ.

For example, $\partial_{\text{set}}(< a, b, c, d, e, b, a, a >) = \{a, b, c, d, e\}$.

Definition 5(Event Log): Let ψ be a set of activities. An event log $L \in \boldsymbol{B}(\psi^*)$ is a multi-set of traces over ψ.

For example, let $\psi = \{a, b, c, d, e\}$ be a set of activities. $s =< a, b, c, e, d >$ is a sequence; $A \in \boldsymbol{B}(\psi)$, $A = \{a^2, b^2, c, d, e^2\}$ is a multi-set; $\sigma =< a, b, c, e >$ is a trace; and $L = \{< a, b, c, e >, < a, b, d, e >\}$ is an event log.

Definition 6 (Petri Net): $PN = (P, T; F, M)$ is a Petri net, where P denotes a finite set of places, T denotes a finite set of transitions, and F is a set of directed arcs from places to transitions or from transitions to places, where

(1) $P \cup T \neq \emptyset$;
(2) $P \cap T = \emptyset$;
(3) $F \subseteq (P \times T) \cup (T \times P)$;
(4) M: $P \rightarrow N$ is a marking function, where for $\forall p \in P$, $M(p)$ denotes the number of tokens in p, and N represents a set of non-negative integers; and
(5) $dom\,(F) \cup cod\,(F) = P \cup T$, where

$dom\,(F) = \{x \in P \cup T | \exists y \in P \cup T (x, y) \in F\}$, $cod\,(F) = \{x \in P \cup T | \exists y \in P \cup T (y, x) \in F\}$.

Definition 7 (Pre-Sets and Post-Sets): Let $PN = (P, T; F, M)$ be a Petri net. For $x \in P \cup T$, we have

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

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

where ${}^\bullet x$ and $x^\bullet$ are called a pre-set and a post-set of x respectively, and ${}^\bullet x \cup x^\bullet$ represents the extension of x.

Definition 8: Let PN be a Petri net. It has the following transition firing rules:

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

(2) If $t \in T$ and $M[t\rangle$, t can be fired and a new marking M' is generated, denoted by $M[t\rangle M'$, where

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

A Petri net can be used as a process model to describe real processes. The process model usually has four basic structures as shown in Figure 4: (a) sequential, (b) choice, (c) parallel, and (d) loop. A model usually contains more than one structure such as those in (e) and (f).

Definition 9 (Firing Sequence): Let $PN = (P, T; F, M)$ be a Petri net. A sequence $s \in T^*$ is a firing sequence, where

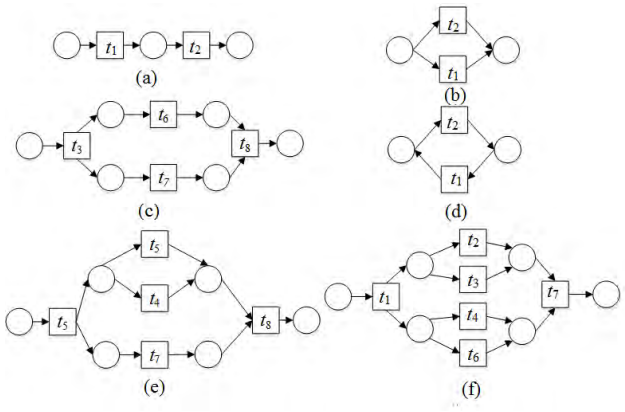


FIGURE 4. Four basic structures and two Petri net models: (a) sequential structure; (b) choice structure; (c) parallel structure; (d) loop structure; (e) a model with a parallel structure containing a choice structure; and (f) a model with a parallel structure containing choice and loop structures.

$\exists M_1$-M_n and $\exists t_1$-$t_n \in T$, for $s =< t_1, t_2, \ldots, t_n >$ and $i \in \{0, 1, \ldots, n-1\}$, $\exists M_i[t_{i+1}\rangle$ and $M_i[t_{i+1}\rangle M_{i+1}$.

Definition 10 (Ordering Relation): Let L be an event log and $\sigma \in L$ be a trace. For $\forall a, b \in \sigma$, four ordering relations between a and b are defined as follows:

(1) Direct-follow relation: it is denoted by $a >_L b$ if there is a trace $\sigma =< t_1, t_2, t_3, t_4, \ldots, t_n >$ and $i \in \{1, 2, 3, \ldots, n-1\}$ such that $\sigma \in L$, $t_i = a$, and $t_{i+1} = b$.

(2) Causal relation: it is denoted by $a \rightarrow_L b$ if $a >_L b$ and $b \not> _L a$.

(3) Parallel relation: it is denoted by $a||_L b$ if $a >_L b$ and $b >_L a$.

(4) Exclusive relation: it is denoted by $a\#_L b$ if $a \not> _L b$ and $b \not> _L a$.

For example, $L = \{\sigma_1 =< a, b, c, d, e >, \sigma_2 =< a, c, b, d, e >\}$ is an event log. The direct-follow relations are $a >_L b$, $b >_L c$, $c >_L d$, $d >_L e$, $a >_L c$, $c >_L b$, and $b >_L d$; the causal relations are $a \rightarrow_L b$, $a \rightarrow_L c$, $b \rightarrow_L d$, $c \rightarrow_L d$, and $d \rightarrow_L e$; the parallel relation is $b||_L c$; and the exclusive relations are $a\#_L a$, $b\#_L b$, $c\#_L c$, $d\#_L d$, $e\#_L e$, $a\#_L d$, $a\#_L e$, $c\#_L e$, and $b\#_L e$.

Definition 11 (α Algorithm): Let L be an event log. α algorithm is defined as follows [3]:

(1) $T_L = \{t \in T | \exists_{\sigma \in L} t \in \sigma\}$

(2) $T_I = \{t \in T | \exists_{\sigma \in L} t = first(\sigma)\}$

(3) $T_O = \{t \in T | \exists_{\sigma \in L} t = last(\sigma)\}$

(4) $X_L = \{(A, B) | A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset$

$\forall_{a \in A} \forall_{b \in B} a \rightarrow_L b \wedge \forall_{a_1, a_2 \in A} a_1 \#_L a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_L b_2\}$

(5) $Y_L = \{(A, B) \in X_L | \forall_{(A', B') \in X_L} A \subseteq A' \wedge B \subseteq B' \Rightarrow$

$(A, B) = (A', B')\}$

(6) $P_L = \{p_{(A,B)} | (A, B) \in Y_L\} \cup \{i_L, o_L\}$

(7) $F_L = \{(a, p_{(A,B)}) | (A, B) \in Y_L \wedge a \in A\} \cup \{(p_{(A,B)}, b)$

$|(A, B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) | t \in T_I\} \cup \{(t, o_L) | t \in T_O\}$

(8) $\alpha(L) = (P_L, T_L, F_L)$.

α algorithm is one of the earliest process mining algorithms. Now, α algorithm is still used to obtain a model.

Definition 12 (Indirect Dependency): Let ψ be a set of activities. For $\forall \sigma \in L$, $a = \sigma[i]$, $b = \sigma[j] \in \psi$, and $i < j$, an indirect dependency between a and b is denoted as $a \Theta b$, where $a^\bullet \cup^\bullet b = \emptyset$ and if $a \in \sigma$, then $\exists b \in \sigma$.

III. INDIRECT DEPENDENCY

This section presents a method to obtain a model with indirect dependencies. Firstly, loop and choice structures are identified. Then association rules are obtained. Finally, an extended Petri net is designed to model the process.

A. LOOP STRUCTURE

In this subsection, activities in loop structures are identified. Then a sequence is formed based on the direct-follow relation of activities.

Definition 13: Let L be an event log, $\sigma \in L$ be a trace, and s be a sequence. For $\forall a \in \sigma$, *sum* (a, σ) indicates the number of a in σ, and *sum* (s, σ) indicates the number of s in σ.

For example, if $\sigma_1 =< e, a, c, c, f >$ is a trace, then *sum* $(a, \sigma_1) = 1$ and *sum* $(c, \sigma_1) = 2$. For $s =< a, c >$, *sum* $(s, \sigma) = 1$.

Definition 14 (Loop Activity): Let L be an event log and $\sigma \in L$ be a trace. $a \in \sigma$ is called a loop activity if *sum* $(a, \sigma) > 1$. The set of all loop activities is denoted as L_{AS}, where $L_{AS} = \{a \in \sigma | \exists \sigma \in L \wedge sum(a, \sigma) > 1\}$.

For example, $\sigma_1 =< e, a, c, c, f >$ and *sum* $(c, \sigma_1) = 2$, so c is a loop activity.

Definition 15 (Loop): Let L_{AS} be a set of loop activities. $\rho \in L^*_{AS}$ is a loop, where for $\forall \rho_i \in L_{AS}$, if $|\rho| = 1$, it is called a 1-length loop; if $|\rho| = 2$, where $\rho[0] >_L \rho[1]$; and if $|\rho| > 2$, where $\rho[i] >_L \rho[i+1]$ and $|\rho|\text{-}2 > i \geq 0$. L_S denotes the set of all loops.

Definition 16: Let $a \in \psi$ be an activity. For $\forall \rho_i \in L^*_{AS}$, *joint* (a, ρ_i) indicates a new sequence $< a, \rho_i >$ and *joint* (ρ_i, a) indicates a new sequence $< \rho_i, a >$.

For example, if $\rho_1 =< e, a, c, c, f >$, then *joint* $(a, \rho_i) =< a, e, a, c, c, f >$ and *joint* $(\rho_i, a) =< e, a, c, c, f, a >$.

Now, an algorithm for identifying loops is proposed as follows.

In Algorithm 1, lines 2-8 indicate that all loop activities are found and added into L_{AS}. Then lines 9-23 indicate that all loops are obtained according to *Definition 15*. Line 24 indicates that loops are added into L_S. Finally, line 27 indicates that the algorithm returns a loop set.

Example 1: Let $L_1 = \{\sigma_1 =< a, b, b, b, c, e >, \sigma_2 =< a, b, c, e >\}$. According to *Definition 14*, *sum* $(b, \sigma_1) = 3$ and $L_{AS} = \{b\}$. From Algorithm 1, we can obtain a 1-length loop $\rho =< b >$.

Example 2: $L_2 = \{\sigma_1 =< a, b, c, d, e, f, g >, \sigma_2 =< a, b, c, d, b, c, d, e, f, g >\}$. Firstly, According to *Definition 14*, *sum* $(b, \sigma_2) = 2$, *sum* $(c, \sigma_2) = 2$, and *sum* $(d, \sigma_2) = 2$. Thus, we have $L_{AS} = \{c, d, b\}$. Then $L_{AS} = L_{AS}$-c, $\rho =< c >$, so $L_{AS} = \{b, d\}$. Since $b >_L c$ and $c = \rho[0]$, we have $L_{AS} = L_{AS}$-b, $\rho =< b, c >$, and $L_{AS} = \{d\}$. Then since

Algorithm 1 Loops

Input: L
Output: L_S
1: $L_{AS} = \emptyset$; $\rho = \varepsilon$;
2: **for** each $\sigma_i \in L$ **do**
3: **for** each $a_j \in \sigma_i$ **do**
4: **if** $sum\ (a_j, \sigma_i) > 1$ **then**
5: $L_{AS} = L_{AS} \cup a_j$;
6: **end if**
7: **end for**
8: **end for**
9: **for** each $a_k \in L_{AS}$ **do**
10: L_{AS}-a_k;
11: $\rho = joint\ (a_k, \varepsilon)$
12: **for** $(i = 0; i < |L_{AS}|; i++)$ **do**
13: **for** $\forall b_m, c_n \in L_{AS}$ **do**
14: **if** $(b_m >_L c_n$ and $c_n = \rho[0]$ and $b_m \in L_{AS})$ **then**
15: $\rho = joint\ (b_m, \rho)$;
16: $L_{AS} = L_{AS}$-b_m;
17: **end if**
18: **if** $(b_m >_L c_n$ and $b_m = \rho[|\rho| - 1]$ and $c_n \in L_{AS})$ **then**
19: $\rho = joint\ (\rho, c_n)$;
20: $L_{AS} = L_{AS}$-c_n;
21: **end if**
22: **end for**
23: **end for**
24: $L_S = L_S \cup \rho$;
25: $\rho = \varepsilon$;
26: **end for**
27: **return** L_S

$\rho[|\rho| - 1] = c$ and $c >_L d$, we have $\rho = joint\ (\rho, d)$ and $L_{AS} = (L_{AS}$-$d) = \emptyset$. Thus, we have $\rho =< b, c, d >$.

B. CHOICE STRUCTURE

In this subsection, we consider a choice structure that only contains a sequential structure. A choice-branch refers to a sequence of activities that satisfy the causal relation in a choice structure. If there is an indirect dependency between the activities of the choice-branch and activities of other discontinuous structures, we use the first activity of each choice-branch to find the indirect dependence between choice activities and other activities. The first activity of choice-branch is called a branch point. As shown in Figure 5, $< b, i >$ is a choice branch. Because b and i belong to the same choice-branch, if there is an indirect dependency between $< b, i >$ and h, we use b to find the indirect dependence between activities of choice-branch and other structures.

Definition 17 (Choice Activity): Let L be an event log, and $\sigma_i, \sigma_j \in L$ be two traces. a is called a choice activity, where $a \in \sigma_i$ and $a \notin \sigma_j$. The set of all choice activities is denoted as

$$C = \{a \in \sigma_2, a \notin \sigma_1 | \exists_{\sigma_1, \sigma_2 \in L} sum(a, \sigma_1) = 0 \wedge sum(a, \sigma_2) = 1\}.$$

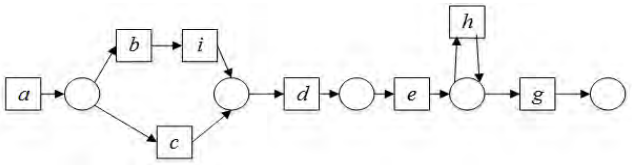


FIGURE 5. A model that contains choice and loop structures.

For example, $L = \{\sigma_1 =< a, b, d, e >, \sigma_2 =< a, c, d, e >\}$. Since $b \in \sigma_1$, $b \notin \sigma_2$, $c \in \sigma_2$, and $c \notin \sigma_1$, we have that b and c are choice activities and $C = \{b, c\}$.

Definition 18 (Branch Point): Let C be a set and $\forall a_i, a_j \in C$ be activities. $a_i \in C$ is called a branch point, where $a_i, a_j \notin L_{AS}$, $a_i \#_L a_j$, and there is no activity $a_k \in C$ such that $a_i \rightarrow_L a_k$. C_{BS} denotes the set of all branch points.

Theorem 1: Let $\sigma \in L$ be a trace and a be an activity. If $a \in L$ and $a \notin \sigma$, then a is a choice activity.

Proof: Because $\forall a \in L$, then $\exists \sigma_i \in L$ and $a \in \sigma_i$. Since $a \in \sigma_i$ and $a \notin \sigma$, a is a choice activity according to *Definition 17*. Thus, the conclusion holds.

According to *Theorem 1* and *Definition 18*, an algorithm for identifying branch points is proposed as follows.

Algorithm 2 Branch Points

Input: L, L_{AS}
Output: C_{BS}
1: $C = \emptyset$, $C_{BS} = \emptyset$;
2: **for** each $\sigma_i \in L$ **do**
3: $C = C \cup (\psi - \partial_{\text{set}}(\sigma_i))$;
4: **end for**
5: $C_{BS} = C$;
6: **for** each $c_i \in C$ **do**
7: **for** each $c_j \in C$ **do**
8: **if** $(c_i \rightarrow_L c_j)$ **then**
9: $C_{BS} = C_{BS}$-c_j;
10: **end if**
11: **if** $(c_j \rightarrow_L c_i)$ **then**
12: $C_{BS} = C_{BS}$-c_i;
13: **end if**
14: **end for**
15: **end for**
16: $C_{BS} = C_{BS}$-L_{AS};
17: **return** C_{BS}

In Algorithm 2, lines 2-4 indicate that according to *Definitions 17*, if $\exists a \in L$ and $a \notin \sigma_i$ then $C = C \cup a$. Next, lines 5-15 indicate that for $\forall a, b$, if $a \rightarrow_L b$, then b is removed from the C_{BS} according to *Definition 18*. Then line 16 indicates that if an activity is both a choice activity and a loop activity, then it is not a branch point. Through the above operations, we can find all branch points. Finally, line 17 returns C_{BS}.

For example, $L = \{\sigma_1 =< a, b, b, c, d, f >, \sigma_2 =< a, c, e, f >\}$ and $\psi = \{a, b, c, d, e, f\}$. For σ_1, $\partial_{set}(\sigma_1) = \{a, b, c, d, f\}$ and ψ-$\partial_{set}(\sigma_1) = \{e\}$, and according to *Theorem 1*, $C = \{e\}$. Then for σ_2, $\partial_{set}(\sigma_2) = \{a, c, e, f\}$. According to *Theorem 1*, $C = C \cup \psi$-$\partial_{set}(\sigma_2)$ and ψ-$\partial_{set}(\sigma_2) = \{b, d\}$. Thus, we have $C = \{b, d, e\}$. Since $b \in L_{AS}$, according to *Definition 18*, $C_{BS} = C_{BS}$-L_{AS}. Thus, we have $C_{BS} = \{d, e\}$.

C. ASSOCIATION RULES

In this subsection, we mine indirect dependencies of activities in loop and choice structures and use association rules to describe indirect dependencies.

Definition 19: Let σ be a trace and $\rho \in \sigma$ be a loop. A loop count is denoted as $sum\ (\rho, \sigma)$.

For example, if $\sigma =< a, b, c, d, e, f, g, e, f, g, e, f, g, h >$ and $\rho =< e, f, g >$, then $sum\ (\rho, \sigma) = 3$.

Definition 20 (Association Tuple): Let $\boldsymbol{B}_1$,$\boldsymbol{B}_2$, $\boldsymbol{B}_3$, and $\boldsymbol{B}$ be multi-sets, where $\boldsymbol{B}_1 = L_S \times C_{BS}$, $\boldsymbol{B}_2 = C_{BS} \times L_S$, $\boldsymbol{B}_3 = C_{BS} \times C_{BS}$, and $\boldsymbol{B} = \boldsymbol{B}_1 \cup \boldsymbol{B}_2 \cup \boldsymbol{B}_3$. An association tuple is denoted as $\tau =< x^{sum(x,\sigma)}, y^{sum(y,\sigma)} > \in \boldsymbol{B}$, where $x^\bullet \cup^\bullet y = \emptyset$, $x, y \in \sigma$, x is the pre-set of the association tuple, y is the post-set of the association tuple, $\boldsymbol{B}\ (\tau) \in \{1, 2, 3, \ldots, n\}$ is the weight of τ, $\tau(x) = x^{sum(x,\sigma)}$, and $\tau(y) = y^{sum(y,\sigma)}$.

For example, $\tau_1 =<< a, b, c >^3, e^1 >$ indicates that e occurs after the loop $< a, b, c >$ occurs 3 times. $\tau_2 =< d^1, < a, b, c >^2 >$ indicates that after d occurs, the loop $< a, b, c >$ occurs twice. $\tau_3 =< d^1, e^1 >$ indicates that after the choice activity d occurs, the choice activity e occurs. For $\tau_1 =<< a, b, c >^3, e^1 >$, we have $\tau_1(x) =< a, b, c, a, b, c, a, b, c >=< a, b, c >^3$ and $\tau_1(y) = e^1 = e$.

Definition 21 (Sub-Trace): Let ψ be a set of activities and $\sigma =< a_0, a_1, a_2 \ldots a_n >$ be a trace. $\gamma =< a_i, a_{i+1}, a_{i+2}, \ldots, a_j >$ is a sub-trace of σ where $0 < i < j < n$. Y denotes the set of all sub-traces of σ.

For example, $\sigma_1 =< e, a, c, c, f >$ is a trace and $\gamma =< a, c >$ is a sub-trace of σ_1.

Definition 22: Let L be an event log. $\sigma \in L$, $a \in C_{BS}$, and $\rho \in L_S$. For $\rho \in \sigma$, $\sigma =< \gamma_1, \rho, \gamma_2, \rho, \ldots, \gamma_n >$, $cut\ (\sigma, \rho) = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$, where $\gamma_i \in Y$ and $0 < i \leq n$. For $a \in \sigma$, $\sigma =< \gamma, a, \gamma' >$, $cut\ (\sigma, a) = \{\gamma, \gamma'\}$, where γ, $\gamma' \in Y$.

For example, $\sigma =< a, b, c, d, e >$ and $cut\ (\sigma, c) = \{\gamma =< a, b >, \gamma' =< d, e >\}$. If $\rho =< b, c >$, then $cut\ (\sigma, \rho) = \{\gamma_1 =< a >, \gamma_2 =< d, e >\}$.

Theorem 2: For $\forall\sigma \in L$, $\forall\rho_i, \rho_j \in L_S$, $\forall a_i \in \rho_i$, and $\forall a_j \in \rho_j$, if $\rho_i \in L$, $a_j \#_L a_i$, and $Y = cut\ (\sigma, \rho_i)$, then $|Y| - 1 = sum\ (\rho_i, \sigma)$.

Proof: Because $\forall a_i \in \rho_i$ and $\forall a_j \in \rho_j$, we have $a_j \#_L a_i$ and $\sigma =<\ldots, a, b, \rho_i, \rho_j, \ldots, \rho_i, \rho_j, d, e, \ldots> \notin L$. For $sum\ (\rho_i, \sigma) = 1$, we have $\forall\sigma =<\ldots, a, b, \rho_i, d, e, \ldots>$, $Y = cut\ (\sigma, \rho_i) = \{\gamma_1 =<\ldots, a, b >, \gamma_2 =< d, e, \ldots>\}$, where $|Y| = 2$, so $|Y| - 1 = 1 = sum\ (\rho_i, \sigma)$. For $sum\ (\rho_i, \sigma) = n \geq 1$, we have $\forall\sigma =<\ldots, a, b, \rho_i, \rho_i, \ldots, \rho_i, d, e, \ldots>$ and $Y = cut\ (\sigma, \rho_i) = \{\gamma_1 =<\ldots, a, b >, \gamma_2 = \varepsilon, \gamma_3 = \varepsilon, \ldots, \gamma_n = \varepsilon, \gamma_{n+1} =< d, e, \ldots>\}$. There are γ_1, γ_{n+1} and $n - 1$ ε in the set Y, where $|Y| - 1 = n = sum\ (\rho_i, \sigma)$. Thus, the theorem holds.

Notice that, the existing algorithms do not consider indirect dependencies between loop and choice activities, they cannot mine models with loop-count-driven-choice structures. We adopt the idea of cutting traces by a loop in this work. An algorithm for mining loop-count-driven-choice structures is proposed as follows.

Algorithm 3 Loop-Count-Driven-Choice Structures

```
Input: L, L_S
Output: B_1
1: B_1 = ∅, Y = ∅;
2: for each ρ ∈ L_S do
3:   for each σ ∈ L do
4:     if (ρ ∉ σ) then
5:       Continue;
6:     end if
7:     if (ρ ∈ σ) then
8:       Y = cut (σ, ρ);
9:       for γ_i ∈ Y do
10:          for each a ∈ C_BS;
11:            if (a ∈ γ_i) where γ_i ∈ Y then
12:            τ =< ρ^{|Y|-1}, a^{sum(a,σ)} >;
13:            B_1 = B_1 ∪ τ;
14:            Break;
15:          end if
16:        end for
17:      end for
18:    end if
19: end for
20: end for
21: return B_1
```

In Algorithm 3, lines 2-8 indicate that a trace is divided into several sub-traces. Then lines 9-20 find the branch points behind the loop. Next, we group them into an association tuple according to *Definition 20*. In addition, we put all the association tuples into $\boldsymbol{B}_1$. Finally, line 21 returns $\boldsymbol{B}_1$.

For example, $L_1 = \{\sigma_1 =< a, b, c, d, e, f, h >, \sigma_2 =< a, b, c, d, b, c, d, e, g, h >\}$. Through Algorithms 1 and 2, we can obtain that $L_S = \{\rho =< b, c, d >\}$ and $C_{BS} = \{f, g\}$. First, according to *Definition 22*, $cut\ (\sigma_1, \rho) = \{\gamma_1 =< a >, \gamma_2 =< e, f, h >\}$. Since $f \in \gamma_2, f \in C_{BS}$, and $sum\ (\rho, \sigma_1) = 1$, we have $\boldsymbol{B}_1 = \{\tau =<< b, c, d >^1, f^1 >\}$. Similarly, we have $\boldsymbol{B}_1 = \{<< b, c, d >^1, f^1 >$ and $<< b, c, d >^2, g^1 >\}$.

In addition, when there are indirect dependencies of activities in choice and loop structures, we call the loop structure whose loop count is affected by a choice activity as a choice-driven-loop structure. Figure 5 shows a model, where b, i, and c are activities in a choice structure and h is a 1-length loop. If the loop count of h is affected by the activities in the choice structure, we call the loop structure a choice-driven-loop structure.

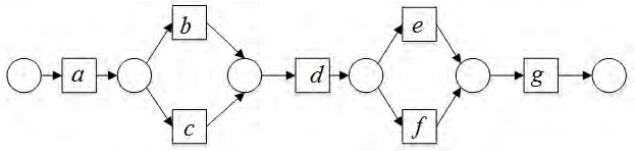


FIGURE 6. A model that contains two choice structures.

We adopt the idea of cutting traces by branch points. An algorithm for mining choice-driven-loop structures is proposed as follows.

Algorithm 4 Choice-Driven-Loop Structures

Input: L, L_S
Output: $\boldsymbol{B}_2$
1: $\boldsymbol{B}_2 = \emptyset$, $Y = \emptyset$;
2: **for each** $a[i] \in C_{BS}$ where $|C_{BS}| > i \geq 0$ **do**
3: **for each**$\sigma \in L$
4: **if** $(a \notin \sigma)$ **then**
5: **Continue**;
6: **end if**
7: **if** $(a \in \sigma)$ **then**
8: $Y = cut\ (\sigma, a)$;
9: **for** each $\rho \in L_S$ **do**
10: **if** $(\rho_j \in \gamma\prime)$ where $\gamma\prime \in Y$ **then**
11: $Y = \text{cut}(\gamma\prime, \rho)$;
12: $\tau =< a^{sum(a,\gamma')}, \rho^{|Y|-1} >$;
13: $\boldsymbol{B}_2 = \boldsymbol{B}_2 \cup \tau$;
14: **break**;
15: **end if**
16: **end for**
17: **end if**
18: **end for**
19: **end for**
20: **return** $\boldsymbol{B}_2$

In Algorithm 4, lines 2-8 are to cut the trace into two sub-traces by branch points in C_{BS}. Then lines 9-16 find the loop behind the choice and cut the sub-trace $\gamma\prime$. In addition, the loop count is obtained according to *Theorem 2*. Next, we group them into an association tuple according to *Definition 20*. Finally, we put all association tuples into $\boldsymbol{B}_2$. Line 20 returns $\boldsymbol{B}_2$.

For example, $L_2 = \{\sigma_1 =< a, m, b, c, d, e, h >$ and $\sigma_2 =< a, n, b, c, d, b, c, d, e, h >^2\}$, where $L_S = \{\rho =< b, c, d >\}$ and $C_{BS} = \{n, m\}$. First, according to *Definition 22*, *cut* $(\sigma_1, m) = \{\gamma =< a >$ and $\gamma\prime =< b, c, d, e, h >\}$. Since $< b, c, d >\in \gamma\prime$ and according to *Theorem 2*, *sum* $(\rho, \gamma\prime) = 1$. Next, we can obtain that $\boldsymbol{B}_2 = \{< m^1, < b, c, d >^1 >\}$. Similarly, we have $\boldsymbol{B}_2 = \{< m^1, < b, c, d >^1 >, < n^1, < b, c, d >^2 >^2\}$.

The indirect dependencies of choice activities are common in real processes. Figure 6 shows a model that contains two choice structures. If b occurs, then only e in another choice structure occurs; and if c occurs, then only f in another choice structure occurs. The aforementioned case is an example of a non-free-choice structure. σ^{++} and ILP algorithms can mine it by adding additional places. Another situation is that if b occurs, then only e in another choice structure occurs; and if c occurs, then e or f in another choice structure occurs. This situation is different from the non-free-choice structure. We call it a semi-non-free-choice structure. The existing algorithms cannot mine the structure.

The indirect dependencies of choice activities can be found in non-free-choice structures and semi-non-free-choice structures. An algorithm for mining above two structures is proposed as follows.

Algorithm 5 Non-Free-Choice and Semi-Non-Free-Choice Structures

Input: L, L_S
Output: $\boldsymbol{B}_3$
1: $\boldsymbol{B}_3 = \emptyset$, $Y = \emptyset$;
2: **for each**$a \in C_{BS}$ **do**
3: **for each**$\sigma \in L$
4: **if** $(a \notin \sigma)$ **then**
5: **Continue**;
6: **end if**
7: **if** $(a[i] \in \sigma)$ **then**
8: $Y = cut\ (\sigma, a[i])$;
9: **for** $b \in C_{BS}$ **do**
10: **if** $(b \in \gamma\prime)$ where $\gamma\prime \in Y$ **then**
11: $\tau =< a^{sum(a,\sigma)}, b^{sum(b,\sigma)} >$;
12: $\boldsymbol{B}_3 = \boldsymbol{B}_3 \cup \tau$;
13: **break**;
14: **end if**
15: **end for**
16: **end if**
17: **end for**
18: **end for**
19: **return** $\boldsymbol{B}_3$

In Algorithm 5, lines 2-8 cut the traces into two sub-traces by branch points in C_{BS}. Then lines 9-10 find the first branch point in $\gamma\prime$. Next, lines 11 and 12 indicate that two branch points are grouped into an association tuple according to *Definition 20*. Finally, line 19 returns $\boldsymbol{B}_3$.

For example, $L_3 = \{\sigma_1 =< a, m, b, e, h >, \sigma_2 =< a, n, b, d, h >^2, \sigma_3 =< a, m, b, d, h >\}$, where $C_{BS} = \{m, n, d, e\}$. First, since $m \in \sigma_1$, we can obtain that *cut* $(\sigma_1, m) = \{\gamma =< a >, \gamma\prime =< b, e, h >\}$. Since $e \in \gamma\prime$, we can obtain that $\boldsymbol{B}_3 = \{< m^1, e^1 >\}$. Then according to *Definition 22*, *cut* $(\sigma_1, e) = \{\gamma =< a, m, b >, \gamma\prime =< h >\}$. Since there is no branch points in $\gamma\prime$, there exists no association tuple. We can analyze σ_2 and σ_3 in the same way. Finally, we have $\boldsymbol{B}_3 = \{< m^1, e^1 >, < m^1, d^1 >, < n^1, d^1 >^2\}$.

Association rules are a common way of describing "if..., then...". It is usually denoted as a form of $X \Rightarrow Y$. In this part, the pre-set and post-set of an association tuple τ can form an association rule, denoted as $\Re_\tau = x^{sum(x,\sigma)} \Rightarrow y^{sum(y,\sigma)}$. We denote TR as the set of association rules, T_{pre-TR} as the

set of $x^{sum(x,\sigma)}$, and $T_{Post-TR}$ as the set of $y^{sum(y,\sigma)}$ in the following content.

Definition 23: Let L be an event log, $\boldsymbol{B}$ be a multi-set of association tuples, τ be an association tuple, $\Re_\tau \in TR$ be an association rule, and $\boldsymbol{B}(\tau)$ be the weight of τ. *Support* of $\Re_\tau$ is defined and calculated as follows.

$$Support(\Re_\tau) = \frac{B(\tau)}{|L|}$$

For example, $\Re_\tau = a \Rightarrow b$, if *Support* $(\Re_\tau) > 0$, which means there is the case of "if a, then b".

Definition 24: Let $\Re_\tau = x^{sum(x,\sigma)} \Rightarrow y^{sum(y,\sigma)}$ be an association rule, $\tau =< x^{sum(x,\sigma)}, y^{sum(y,\sigma)} >$ be an association tuple, $N_{\tau(x)}$ be the number of $x^{sum(x,\sigma)}$, and $\boldsymbol{B}(\tau)$ be the weight of τ. *Confidence* of $\Re_\tau$ is defined and calculated as follows.

$$Confidence(\Re_\tau) = \frac{\mathbf{B}(\tau)}{N_{\tau(x)}}$$

For example, $\Re_\tau = a \Rightarrow b$, if *confidence* $(\Re_\tau) = 0.8$, which means if a occurs, the probability of b occurring is 0.8.

Theorem 3: For $\forall \Re_\tau = x \Rightarrow y \in TR$, if *Confidence* $(\Re_\tau) = 1$ and *Support* $(\Re_\tau) > 0$, then $x \Theta y$.

Proof: Since *Support* $(\Re_\tau) > 0$ and $\boldsymbol{B}(\tau) > 0$, $\exists \tau =< x, y >$ where $x^\bullet \cup^\bullet y = \emptyset$ and $x, y \in \sigma$. Since *Confidence* $(\Re_\tau) = 1$, $\boldsymbol{B}(\tau) = N_{\tau(x)}$, which means that if $x \in \sigma$, then $\exists y \in \sigma$. Thus, the theorem holds.

Theorem 3 means that we can describe implicit dependencies by association rules with *Confidence* $(\Re_\tau) = 1$ and *Support* $(\Re_\tau) > 0$. According to *Theorem 3* and *Definitions 23* and *24*, an algorithm for mining association rules and calculating the support and confidence of association rules is proposed as follows.

In Algorithm 6, lines 2-6 indicate that the *Support* $(\Re_\tau)$ is calculated according to *Definition 23*. Then lines 7-15 indicate that the *Confidence* $(\Re_\tau)$ is calculated according to *Definition 24*. In addition, lines 16-23 find association rules with *Confidence* $(\Re_\tau) = 1$ and *Support* $(\Re_\tau) > 0$ according to *Theorem 3*. Finally line 24 returns *TR*, T_{pre-TR}, and $T_{post-TR}$.

For example, $L_2 = \{\sigma_1 =< a, m, b, c, d, e, h >$ and $\sigma_2 =< a, n, b, c, d, b, c, d, e, h >^2\}$. According to Algorithm 4, we can get that $\boldsymbol{B}_2 = \{< m^1, < b, c, d >^1 >, < n^1, < b, c, d >^2 > 2\}$ and $TR = \{\Re_{\tau 1} = m^1 \Rightarrow < b, c, d >^1, \Re_{\tau 2} = n^1 \Rightarrow < b, c, d >^2\}$. Then according to *Definitions 23* and *24*, we have:

Support $(\Re_{\tau 1}) = 1/3 = 0.33$, *Confidence* $(\Re_{\tau b1}) = 1/1 = 1$,

Support $(\Re_{\tau 2}) = 2/3 = 0.67$, and *Confidence* $(\Re_{\tau 2}) = 2/2 = 1$.

According to *Theorem 3*, we have $TR = \{\Re\tau_1 = m^1 \Rightarrow < b, c, d >^1, \Re_{\tau 2} = n^1 \Rightarrow < b, c, d >^2\}$, $T_{pre-TR} = \{m^1, n^1\}$, and $T_{post-TR} = \{< b, c, d >^1, < b, c, d >^2)\}$. However, for $\boldsymbol{B}_3 = \{\tau_1 =< m^1, e^1 >, \tau_2 =< m^1, d^1 >, \tau_3 =< n^1, d^1 >^2\}$ and $TR = \{\Re_{\tau 3} = m^1 \Rightarrow e^1, \Re_{\tau 4} = m^1 \Rightarrow d^1 >, \Re_{\tau 5} = n^1 \Rightarrow d^1\}$, we can see that if m occurs once, e or d occurs. Since *Confidence* $(\Re_{\tau 3}) = 0.5$ and *Confidence* $(\Re_{\tau 4}) = 0.5$, $\Re_{\tau 3}$ and $\Re_{\tau 4}$ cannot describe indirect dependencies.

Algorithm 6 Association Rules

Input: $\boldsymbol{B} = \boldsymbol{B}_1 \cup \boldsymbol{B}_2 \cup \boldsymbol{B}_3$
Output: *TR*, T_{pre-TR}, $T_{post-TR}$
1: $N_{\tau(x)} = 0$, $TR = \emptyset$, $T_{pre-TR} = \emptyset$, $T_{post-TR} = \emptyset$;
2: **for** each $\tau =< x^{sum(x,\sigma)}, y^{sum(y,\sigma)} > \in \boldsymbol{B}$ **do**
3: $\Re_\tau = x^{sum(x,\sigma)} \Rightarrow y^{sum(y,\sigma)}$;
4: $TR = TR \cup \Re_\tau$;
5: *Support* $(\Re_\tau) = \boldsymbol{B}(\tau)$div $|L|$;
6: **end for**
7: **for** each $\tau =< x^{sum(x,\sigma)}, y^{sum(y,\sigma)} > \in \boldsymbol{B}$ **do**
8: **for** each $\tau_i \in \boldsymbol{B}$ **do**
9: **if** $(\tau(x) = \tau_i(x))$ **then**
10: $N_{\tau(x)} = N_{\tau(x)} + 1$;
11: **end if**
12: **end for**
13: ***Confidence***$(\Re_\tau) = \boldsymbol{B}(\tau)$div $N_{\tau(x)}$;
14: $N_{\tau(x)} = 0$;
15: **end for**
16: **for** each $\Re_\tau \in TR$**do**
17: **if** (*Confidence* $(\Re_\tau) = 1$ **&&** *Support* $(\Re_\tau) > 0$) **then**
18: $T_{pre-TR} = T_{pre-TR} \cup x^{sum(x,\sigma)}$;
19: $T_{post-TR} = T_{post-TR} \cup y^{sum(y,\sigma)}$;
20: **end if**
21: **else then**
22: $TR = TR$-$\Re_\tau$;
23: **end for**
24: **return** *TR*, T_{pre-TR}, $T_{post-TR}$;

D. EXTENDED PETRI NET

In this subsection, we design a new Petri net to build a model with the indirect dependencies.

Definition 25 (Dependency Petri Net): $DPN = (PN, T_{pre-TR}, T_{Post-TR}, TR)$ is a dependency Petri net, where *PN* is a Petri net, T_{pre-TR} denotes $x^{sum}x, \sigma$ of association rules, $T_{Post-TR}$ denotes $y^{sum}y, \sigma$ of association rule, and *TR* denotes a finite set of association rules, where it has the following transition firing rules:

1) For $\forall t_i^1 \Rightarrow t_l^1 \in TR$ and $t_l \in T_{Post-TR}$, if t_i has enabled, and $p \in^\bullet t_l$: $M(p) \geq 1$, then t_l is enabled at M, denoted by $M[t_l\rangle$;

2) For $\forall t_j^1 \Rightarrow < t_o, \ldots, t_p >^n \in TR$, $n \in N_+$, and $< t_o, \ldots, t_p > \in T_{Post-TR}$, if t_j has enabled, and $p \in^\bullet t_o$: $M(p) \geq 1$, then $< t_o, \ldots, t_p >$ is enabled at M, and $< t_o, \ldots, t_p >$ can enable n times, denoted by $M[< t_o, \ldots, t_p >^n\rangle$;

3) For $\forall < t_a, \ldots, t_b >^n \Rightarrow t_k^1 \in TR$, $t_k \in T_{Post-TR}$, if $< t_a, \ldots, t_b >$ has enabled n times, and $p \in^\bullet t_k$: $M(p) \geq 1$, t_k is enabled at M, denoted by as $M[t_k\rangle$; and

4) If $t \notin T_{Post-TR}$, and $t \in T_{pre-TR}$ has not enabled, the firing rules are consistent with *PN*.

Association rules are used to describe indirect dependencies.

For example, Figure 7 is a *DPN* model, where $TR = \{t_2^1 \Rightarrow t_5^1\}$, $T_{pre-TR} = \{t_2^1\}$, $T_{Post-TR} = \{t_5^1\}$. For t_4, $p \in^\bullet t_4$: $M(p) \geq 1$, so t_4 can be fired. For $p \in^\bullet t_5$: $M(p) \geq 1$, t_2 occurs

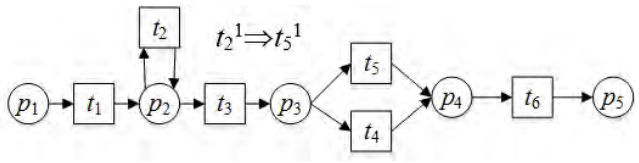


FIGURE 7. A *DPN* Model with Indirect Dependency.

once, and then t_5 can be fired. In addition, if t_2 has occurred once, t_4 cannot be fired.

E. α^{TR} ALGORITHM

Based on α lgorithm, an algorithm is proposed to mine association rules and indirect dependencies as follows.

Definition 26 (α^{TR} Algorithm): Let L is an event log. α^{TR} algorithm is defined as follow:

(1) $T_L = \{t \in T | \exists_{\sigma \in L} t \in \sigma\}$

(2) $T_I = \{t \in T | \exists_{\sigma \in L} t = first(\sigma)\}$

(3) $T_O = \{t \in T | \exists_{\sigma \in L} t = last(\sigma)\}$

(4)$L_{AS} = \{a | \exists_{\sigma \in L} sum(a, \sigma) > 1\}$

(5) $C = \{a | \exists_{\sigma_i, \sigma_j \in L} sum(a, \sigma_i) = 0 \wedge sum(a, \sigma_j) = 1\}$

(6) $L_S = \{\rho | \forall \rho[i] \in L_{AS}\}$

(7) $C_{BS} = \{a, b | \exists_{a,b \in C} a \#_L b\}$

(8) $X_L = \{(A, B) | A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \forall_{a \in A} \forall_{b \in B} a \rightarrow_L b \wedge \forall_{a_1, a_2 \in A} a_1 \#_L a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_L b_2\}$

(9) $Y_L = \{(A, B) \in X_L | \forall_{(A', B') \in X_L} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B')\}$

(10) $P_L = \{p_{(A,B)} | (A, B) \in Y_L\} \cup \{i_L, o_L\}$

(11) $F_L = \{(a, p_{(A,B)}) | (A, B) \in Y_L \wedge a \in A\} \cup \{(p_{(A,B)}, b) | (A, B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) | t \in T_I\} \cup \{(t, o_L) | t \in T_O\}$

(12) $T_{pre-TR} = \{x^{sum(x,\sigma)} | x \in (C_{BS} \cup L_S)\}$

(13) $T_{post-TR} = \{y^{sum(y,\sigma)} | y \in (C_{BS} \cup L_S)\}$

(14) $TR = \{x^{sum(x,\sigma)} => y^{sum(y,\sigma)} | x^{sum(x,\sigma)}, y^{sum(y,\sigma)} \in (T_{pre-TR} \cup T_{post-TR}) \wedge Support(\Re_\tau) > 0 \wedge confidence(\Re_\tau) = 1\}$

(15) $\alpha^{TR}(L) = (P_L, T_L, F_L, T_{pre-TR}, T_{post-TR}, TR)$

Notice that operations in steps (4)-(7), (12), (13), and (14) are different from α algorithm, where (4) and (6) are according to Algorithm 1; (5) and (7) are obtained by Algorithm 2; (12), (13), and (14) are obtained by Algorithms 3-6. In α^{TR} algorithm, step (1) gets all transitions. Then steps (2) and (3) obtain the first and last transitions, respectively. Steps (4)-(7) aim to find the transitions in loop or choice structure. Steps (8) and (9) mine the causal relations. Then, steps (10) and (11) generate the places and directed arcs, respectively. Steps (12)-(14) get the association rules. Finally, the α^{TR} algorithm returns a *DPN* model. Let $|L_S| = m$, $|L| = k$, $|Y| = i$, $|C_{BS}| = j$, and $|T_L| = n$. Since Algorithm 3 uses four layers of loops, the worst complexity of α^{TR} algorithm is $O(n^4)$. In fact, since m, i, j, $k < n$, the computational complexity of Algorithm 3 is $m \times k \times i \times j$, which is lower than $O(n^4)$. Next, we use an example to further illustrate the A algorithm and its complexity.

For example, $L = \{\sigma_1 =< a, b, c, d, e, f, h >, \sigma_2 =< a, b, c, d, b, c, d, e, g, k, h >\}$. Firstly, in step (1), $T_L = \{a, b, c, d, e, f, g, h, k\}$. Steps (2) and (3) get that $T_I = \{a\}$ and $T_O = \{h\}$. Then by Algorithms 1 and 2, steps (4)-(7) can obtain that $L_{AS} = \{b, c, d\}$, $C = \{g, h, k\}$, $L_S = \{\rho =< b, c, d >\}$, and $C_{BS} = \{g, h\}$. In step (8), we have $X_L = \{(\{a\}, \{b\}), (\{b\}, \{c\}), (\{c\}, \{d\}), (\{d\}, \{e\}), (\{e\}, \{f\}), (\{e\}, \{g\}), (\{g\}, \{k\}), (\{k\},\{h\}), (\{f\}, \{h\})\}$. In step (9), we have $Y_L = \{(\{a\}, \{b\}), (\{b\}, \{c\}), (\{c\}, \{d\}), (\{d\}, \{e\}), (\{e\}, \{f, g\}), (\{g\}, \{k\}), (\{k, f\},\{h\})\}$. In step (10), we can get that $P_L = \{p(\{a\}, \{b\})\}, p(\{b\}, \{c\}), p(\{c\}, \{d\}), p(\{d\}, \{e\}), p(\{e\}, \{f\}, \{g\}), p(\{g\}, \{k\}), p(\{k, f\}, \{h\}), i_L, o_L\}$. Then, in step (11), $F_L = \{(a, p(\{a\}, \{b\})), (p(\{a\}, \{b\}), b), (b, p(\{b\}, \{c\})), (p(\{b\}, \{c\}), c), (c, p(\{c\}, \{d\})), (p(\{c\}, \{d\}), d), (d, p(\{d\}, \{e\})), (p(\{d\}, \{e\}), e), (e, p(\{e\}, \{f\}), g), (p(\{e\}, \{f\}, \{g\}, f), (p(\{e\}, \{f\}, \{g\}, g), g, p(\{g\}, \{k\}), (p(\{g\}, \{k\}, k), (k, p(\{k\}, \{f\}, \{h\}), (f, p(\{k\}, \{f\}, \{h\})), (p(\{k\}, \{f\}, \{h\}), h), (i_L, a), (h, o_L))\}$. Next, through Algorithms 3 and 6, steps (12)-(14) get that $T_{pre-TR} = \{< b, c, d >^1, < b, c, d >^2\}$, $T_{post-TR} = \{g^1, f^1\}$, and $TR = \{< b, c, d >^1 \Rightarrow f^1 >, < b, c, d >^2 \Rightarrow g^1\}$. Finally, step (13) gets that $\alpha^{TR}(L) = (P_L, T_L, F_L, T_{pre-TR}, T_{post-TR}, TR)$. Since $|L_S| = 1$, $|L| = 2$, $|Y| = 2$, $n = |T_L| = 9$, and $|C_{BS}| = 2$, the computation of Algorithm 3 is $1 \times 2 \times 2 \times 2$.

IV. EXPERIMENTAL EVALUATION

In this section, experiments are conducted based on two artificial cases and two real business processes on ProM [44]. We compare α^{TR} algorithm with α^{++}, HM, ILP, and IM algorithms, and discuss the precision and fitness of the models obtained by the five algorithms. We use the method proposed in [45] to calculate the precision and the tool named *Replay a Log on Petri Net for Conformance Analysis* for testing the fitness values of the models. We name the plugin containing the α^{TR} algorithm as alphaTR indirect dependencies miner. The plugin is publicly accessible at https://github.com/sdust-sunhuiming/miner.

The rest of the experiment is organized as follows. In subsection A, we discuss an artificial model with a semi-non-free-choice structure and the algorithm efficiency. Then in subsection B, an artificial model that contains a choice-driven-loop structure is discussed. Next, subsection C introduces the outpatient process model of a hospital in Qingdao, which contains a non-free-choice structure. In addition, subsection D discusses a model of an e-commerce system that contains a loop-count-driven-choice structure.

A. SEMI-NON-FREE-CHOICE STRUCTURE

In this subsection, the semi-non-free-choice structure is discussed. We first use the following simple artificial event log and mine a process model. $L_1 = \{< a, b, d, e, h >^2, < a, c, d, e, h >, < a, c, d, f, h >^2, < a, c, d, g, h >, < a, i, d, e, h >, < a, i, d, f, h >\}$. Next, we use five algorithms to mine the event log L_1. Finally, the precision and fitness of the model are calculated.

Figure 8 is a model obtained by α^{++} algorithm. We found that models obtained by ILP and IM algorithms are the same as the one in Figure 8. Figure 9 is a model obtained by HM algorithm. It is a c-net. We can convert it into a Petri net which is also the same as the model in Figure 8. The event log is a record of the process. We can see that from the L_1, if b occurs in the first choice structure, then only e occurs in the second choice structure, and no other activities in the second choice structure occur after b. However, the models obtained by α^{++}, HM, ILP, and IM algorithms allow traces that do not exist in L_1. It means that there are behaviors in the model that do not exist in the business process. For example, $< a, b, d, f, h >$ and $< a, b, d, g, h >$ are two traces that do not exist in L_1, but they are allowed by the model in Figures 8 and 9.

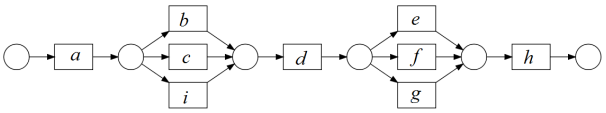


FIGURE 8. A model mined by α^{++} algorithm.

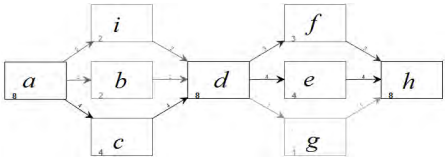


FIGURE 9. A model mined by HM algorithm.

Figure 10 is a *DPN* model obtained by α^{TR} algorithm. Compared with the models obtained by other algorithms, we can see that it has an association rule $b^1 \Rightarrow e^1$. Thus, the model does not allow f and g to occur after b, which reduces the possibility of traces that do not exist in the process. There is no association rule on c, so activities e, g, and f may occur after c.

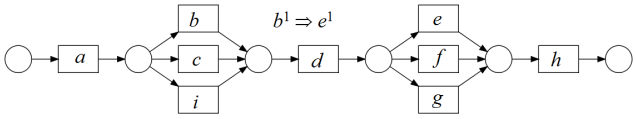


FIGURE 10. A model mined by α^{TR} algorithm.

Figure 11 shows the precision of the five models. The precision is calculated by the *1-alignment* method proposed in [45]. The precision of models obtained by α^{++}, HM, ILP, and IM algorithms is 0.9167, the precision of α^{TR} mining model is 0.9706. Figure 12 shows the fitness of the five models. The fitness of the five models is 1. It means that our algorithm improves the precision of the model while maintaining the same degree of fitness as other algorithms.

The complexity of the α^{++} algorithm is exponential in the number of activities [5]. When the event log contains a small number of activities, the α^{++} algorithm is more efficient. However, when the number of activities is large,

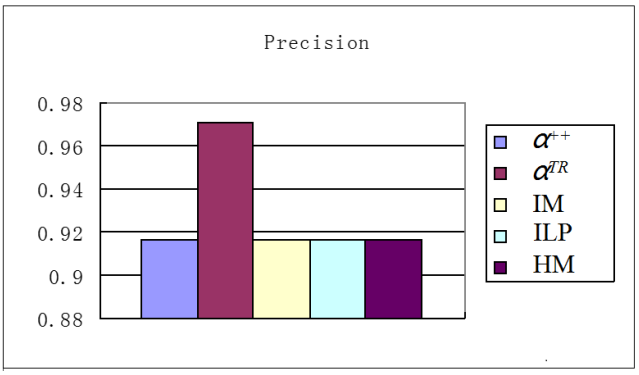


FIGURE 11. Precision.

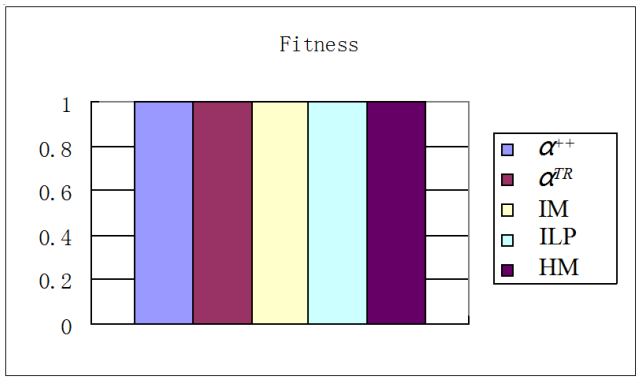


FIGURE 12. Fitness.

the efficiency of the α^{++} algorithm will decrease. The complexity of the ILP algorithm is worst-case exponential [13]. When the event log contains large traces, the efficiency of ILP algorithm is much lower than our algorithm. However, ILP algorithm has an outstanding capability to mine various models with complex structures. Experiments show that HM and IM algorithms take less time than ours. Our approach sacrifices efficiency but gets a model with high precision.

In summary, the α^{TR} algorithm can effectively mine semi-non-free-choice structures.

B. CHOICE-DRIVEN-LOOP STRUCTURE

In this subsection, we discuss a model that contains a choice-driven-loop structure. L_2-L_6 are five different event logs generated by the same process as shown in Table 1.

TABLE 1. The information of five event logs.

Event log	Number of Activities	Number of Events	Number of Traces
L_2	15	4263	255
L_3	15	7518	441
L_4	15	8582	526
L_5	15	18782	1114
L_6	15	34993	2053

Figure 13 shows a *DPN* model obtained by α^{TR} algorithm. We find three association rules in the model. For example, after k occurs, then the loop $< m, n, o >$ will occur three

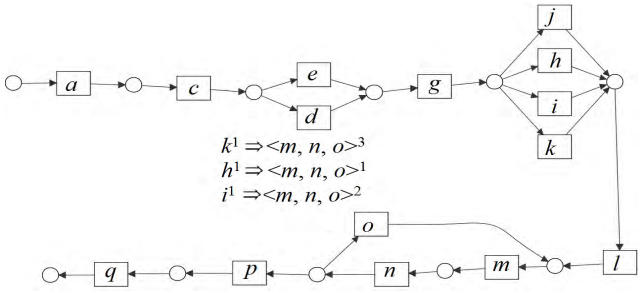


FIGURE 13. A model mined by α^{TR} algorithm.

times, and other times are not allowed. In this way, the possibility of the model generating traces that do not exist in the log is reduced. Our approach can improve the precision of models by using association rules to limit the firing rules of the Petri net.

Figure 14 shows the model obtained by α^{++} algorithm. The models obtained by HM and ILP are the same as the one in Figure 14. The above three algorithms cannot mine choice-driven-loop structures. It means that the loop can occur any number of times after the choice in models obtained by α^{++}, HM, and ILP algorithms. For example, this model allows behaviors like $< a, c, e, g, k, i, m, n, o, m, n, p, q >$, which does not exist in the log. This model has low precision.

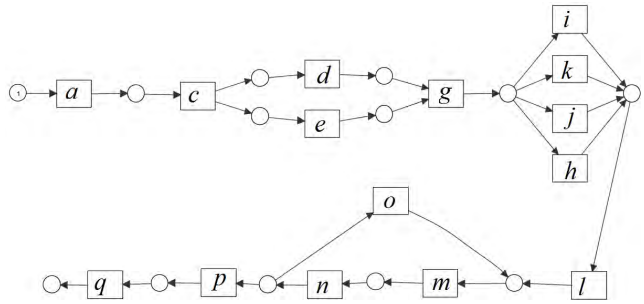


FIGURE 14. A model mined by α^{++}, HM, and ILP algorithms.

Figure 15 is the model obtained by IM algorithm. The algorithm also cannot mine choice-driven-loop structures. This algorithm additionally mines three invisible transitions, which makes the model more complex. Besides, the loop can occur any number of times after the choice structure, which is obviously wrong. Therefore, the model with low precision is complex.

Figure 16 shows the precision of the models. We can see that since we have mined the association rules in the model, the proposed algorithm mining models have the highest precision. The precision of the IM algorithm mining models is lower than that of the α^{TR} algorithm, which is higher than other algorithms. The models of α^{++}, HM, and ILP algorithms mining are the same, so the precision of the models are all the same. However, the precision is lower than the α^{TR} and IM algorithms.

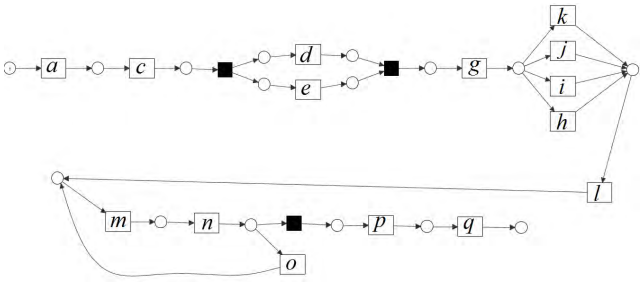


FIGURE 15. A model mined by IM algorithm.

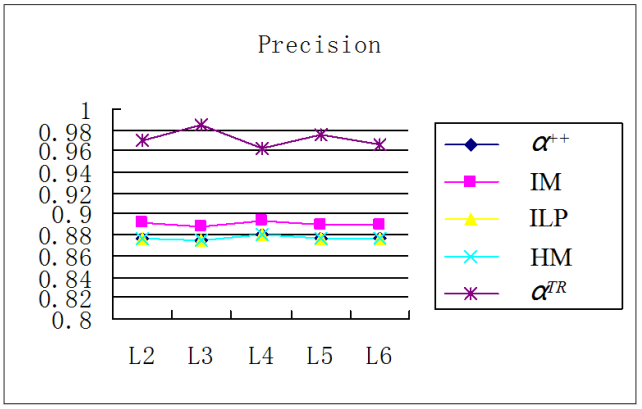


FIGURE 16. Precision.

Figure 17 shows that the fitness of the models obtained by the five algorithms are all 1, which means that all the traces in the event log can be replayed in the models. However, the precision of other algorithms mining model is lower, which means there are more traces that the model can produce, but they do not exist in the real process.

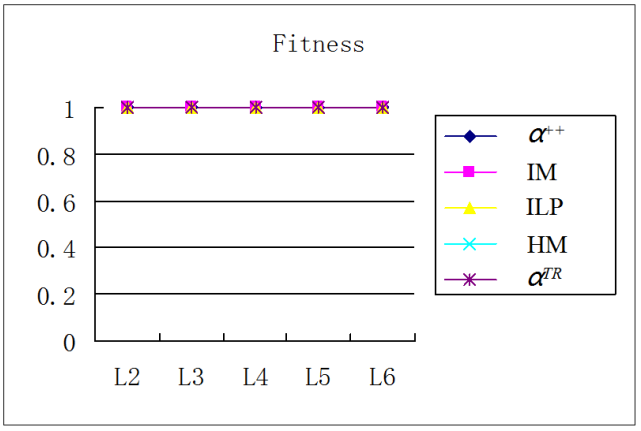


FIGURE 17. Fitness.

In summary, the α^{TR} algorithm can effectively mine the models with choice-driven-loop structures.

C. NON-FREE-CHOICE STRUCTURE

In this subsection, we analyze the non-free-choice structure with a real outpatient process of a hospital in Qingdao. The capability of five algorithms to mine models with a

non-free-choice structure is compared. L_7-L_{11} are five different event logs generated by the aforementioned process. Table 2 shows information about five event logs. Table 3 shows the symbols in the model and the activities represented by the symbols. Next, we compare the models obtained by five algorithms.

TABLE 2. The information of five event logs.

Event log	Number of Activities	Number of Events	Number of Traces
L_7	16	4316	320
L_8	16	7283	534
L_9	16	11278	828
L_{10}	16	21713	1601
L_{11}	16	38986	2944

TABLE 3. Notations.

Symbols	Activities
a	Seeking medical advice
b	Registering a hospital card
c	Entering patient information
d	Reading the hospital card
e	Registering a number
f	Waiting in line in front of the hospital department
g	Calling a number
h	Doctor consultation
i	Giving a new medical record book
j	Reading the medical record book
k	Asking the patient about the condition
l	Giving doctor's advice
m	Updating medical record book
n	Paying by card
o	Executing doctor's advice
p	Leaving hospital

Figure 18 shows the model obtained by α^{++} algorithm. This algorithm adds extra places (two circles marked as blue in Figure 18) between the two choice structures. Since these two places limit the occurrence of activities, it can describe the non-free-choice structure. However, due to the added new places and directed arcs, it reduces the simplicity of the model.

Figure 19 shows a model obtained by HM and ILP algorithms. These two algorithms cannot mine non-free-choice structures in the model. These models allow traces that do not exist in the outpatient process. In addition, if there are fewer activities between the two choice structures, the ILP algorithm can mine non-free-choice structures in a similar way to the α^{++} algorithm. However, when there are many activities between the two choice structures, the ILP algorithm cannot mine non-free-choice structures in the model.

Figure 20 shows the model obtained by IM algorithm. It uses the method of adding invisible transitions to improve the precision, but still does not mine non-free-choice structures in the model, so the model obtained by this algorithm has low precision.

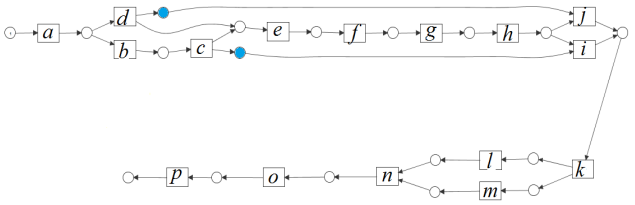


FIGURE 18. A model mined by α^{++} algorithm.

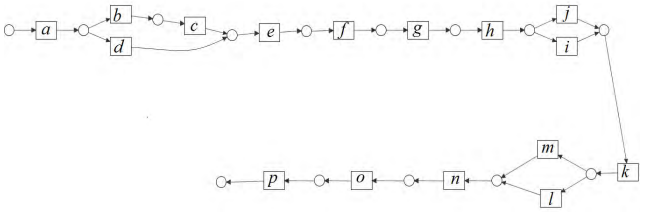


FIGURE 19. A model mined by HM and ILP algorithm.

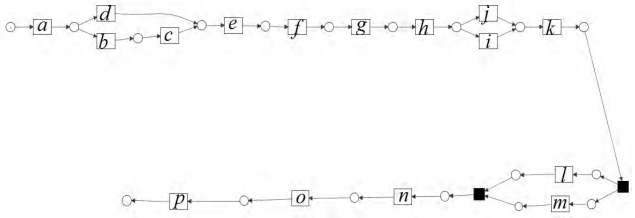


FIGURE 20. A model mined by IM algorithm.

The model obtained by IM, HM, and ILP algorithms cannot accurately reflect the outpatient process. When the patient comes to the hospital for the first time, he/she needs to register a hospital card and then is given a new medical record book. However, those people who are not the first time are called ''old patients''. They have a hospital card and a medical record book previously. The hospital card and medical record book can be only read and updated. The model obtained by IM, HM, and ILP algorithms allow that ''old patients'' are given a new medical record book, which is impossible in the real process. Therefore, the models obtained by IM, HM, and ILP algorithms have low precision.

Figure 21 shows a *DPN* model obtained by α^{TR} algorithm. This algorithm mines the association rules in the model. The association rules limit the firing rules of Petri nets through

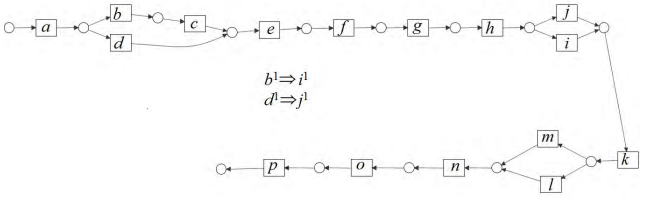


FIGURE 21. A model mined by α^{TR} algorithm.

association rules. It can describe non-free-choice structures in the model. Compared with the IM and α^{++} algorithms, our algorithm does not add additional places, directed arcs, and invisible transitions, which ensures the simplicity of the model. Compared with the model of ILP algorithm and HM algorithm, the model by α^{TR} algorithm has high precision. It does not allow behaviors that do not exist in the outpatient process.

Figures 22 and 23 show the precision and fitness of models obtained by five algorithms. It can be seen from the figure that the fitness and precision of our algorithm mining models are the same as α^{++} algorithm. We adopt association rules on Petri nets. The α^{++} algorithm uses extra places to change the firing rule of the Petri net. It has been proved that the two algorithms achieve the same effect when mining non-free-choice structures. However, from the perspective of model structure, the model by our algorithm mining is simpler. The models obtained by the five algorithms have the same fitness, but the models obtained by α^{++} and α^{TR} algorithms have higher precision than the other three algorithms.

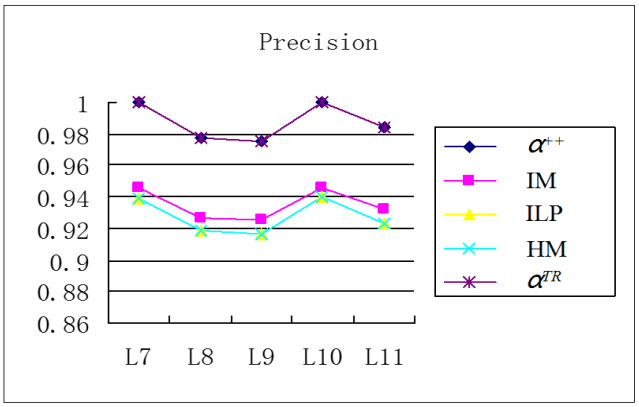


FIGURE 22. Precision.

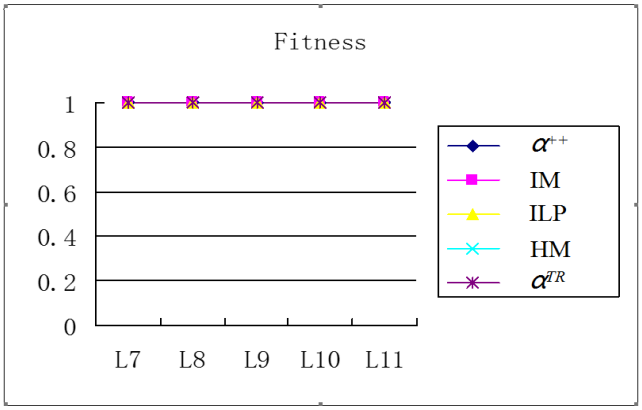


FIGURE 23. Fitness.

In summary, our algorithm can effectively mine models with non-free-choice structures, and the simple models have higher fitness and precision.

D. LOOP-COUNT-DRIVEN-CHOICE STRUCTURE

In this subsection, we analyze the real process of an e-commerce system. In the process, a loop-count-driven-choice structure is included. This kind of structure is widely found in systems that include password verification in login processes. Therefore, studying loop-count-driven-choice structure is beneficial to better describe the system process. Table 4 shows information about five event logs. Table 5 shows the symbols in the model and the activities represented by the symbols. Next, we compare the models obtained by five algorithms.

TABLE 4. The information of five event logs.

Event log	Number of Activities	Number of Events	Number of Traces
L_{12}	14	9048	580
L_{13}	14	13302	829
L_{14}	14	40902	2846
L_{15}	14	67444	4258
L_{16}	14	133380	8550

TABLE 5. Notations.

Symbols	Activities
a	Opening the system
b	Entering the system home page
c	Searching by product name
d	Searching by product category
e	Adding item to cart
f	Submitting the order
g	Paying the order
h	Entering a password
i	Verifying password
j	Paying successfully
k	payment failing
l	Freezing the account
m	Re-entering a password
n	Closing the system

Figure 24 shows a *DPN* model obtained by α^{TR} algorithm. We can see that the sub-process ''re-entering a password and verifying the password, and payment failing'' occurs once or twice without causing the account to be frozen. When there are no ''payment failing'' and ''re-entering a password'' in the trace, which proves that the consumer pays successfully after entering the password once. When the aforementioned

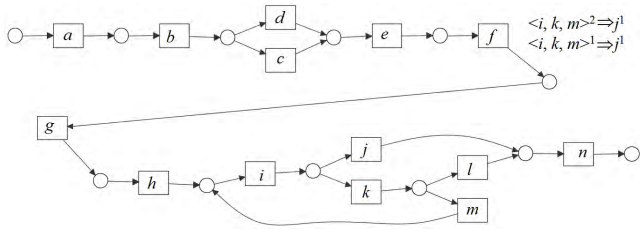


FIGURE 24. A Model Mined by α^{TR} Algorithm.

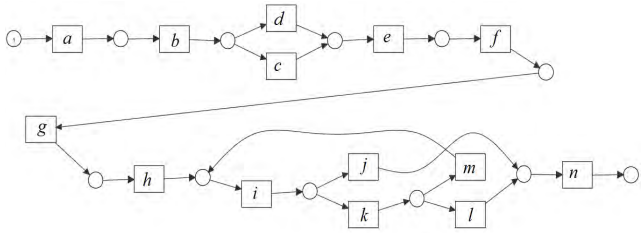


FIGURE 25. A Model Mined by α^{++}, HM, and ILP Algorithm.

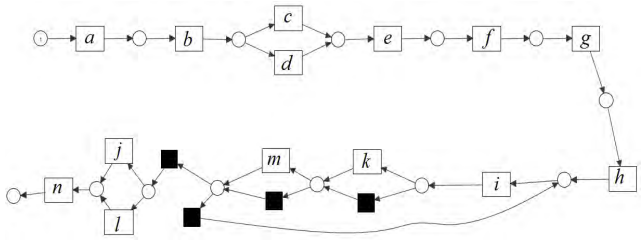


FIGURE 26. A Model Mined by IM Algorithm.

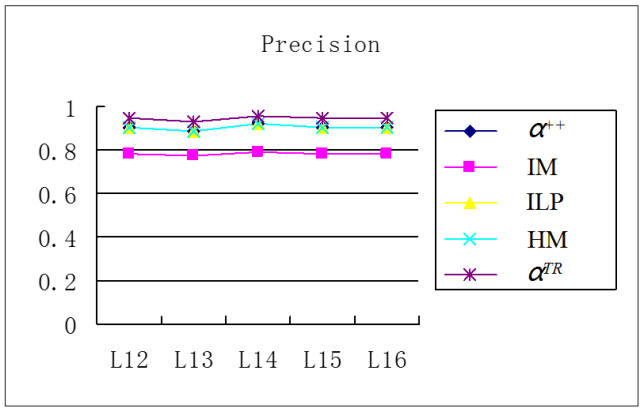


FIGURE 27. Precision.

sub-process occurs three times, which means the consumer has reentered a wrong password three times. When the consumer re-enters the password next time, if the verification still fails, it proves that the account may be at risk. In order to protect the security of the account, it is necessary to temporarily freeze the account. These two association rules obtained by α^{TR} algorithm helps us better understand the freeze account process.

Figure 25 shows the model obtained by α^{++}, ILP, and HM algorithms. The above algorithms cannot obtain loop-count-driven-choice structures in the model. It shows that freezing the account is likely to occur after any number of times of re-entering the password and verification failed, which is obviously wrong. Therefore, this model has low precision.

Figure 26 shows the model obtained by IM algorithm. First, there are several invisible transitions in this model, which reduce the simplicity of the model. In addition, this model allows traces like $<\ldots, k, m, j, \ldots>$, which is obviously not presented in real business. Therefore, IM algorithm cannot mine loop-count-driven-choice structures, and models mined by it has low precision.

Next, we compare the precision and fitness of the model. Figures 27 and 28 show the precision and fitness of models. Although the models obtained by the five algorithms have the same value of fitness, models mined by our algorithm have the highest precision over the others.

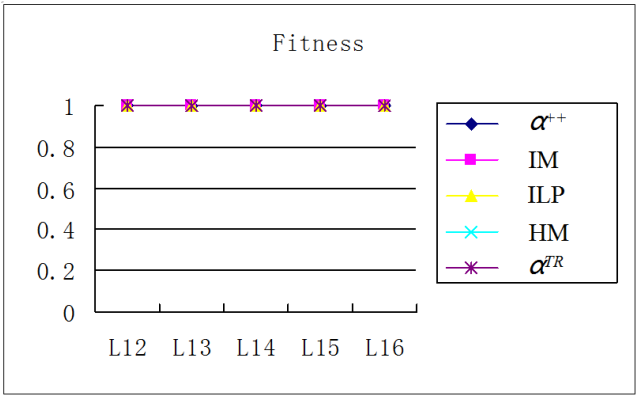


FIGURE 28. Fitness.

In summary, our algorithm can effectively mine the model containing the loop-count-driven-choice structures.

V. CONCLUSION

In this paper, an algorithm is proposed to construct models with indirect dependencies. Through this algorithm, we can mine the four kinds of structures that contain indirect dependency, i.e., loop-count-driven-choice structure, choice-driven-loop structure, non-free-choice structure, and semi-non-free-choice structure. Then in order to solve the problem that Petri nets are difficult to express indirect dependence, we extend Petri nets and express indirect dependencies by association rules. Some experiments on two real-world business process cases and two artificial cases are carried out on ProM. The results show that our algorithm is effective. Besides, models mined by our algorithm have higher precision than other algorithms mine. This method can be used to construct real process models, which helps people to further understand the system operation, thereby discovering system bottlenecks and optimizing the process. Our future work will focus on indirect dependencies in multiple parallel structures and other complex structures.

REFERENCES

[1] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin, Germany: Springer, 2011, pp. 1–10.

[2] A. K. A. de Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters, "Workflow mining: Current status and future directions," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*. Berlin, Germany: Springer, 2003, pp. 389–406.

[3] W. 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.

[4] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters, "Process mining: Extending the α-algorithm to mine short loops," Ph.D. dissertation, Dept. Technol. Man., Eindhoven Univ. Technol., Eindhoven, The Netherlands, 2004.

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

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

[7] J. Li, D. Liu, and B. Yang, "Process mining: Extending α-algorithm to mine duplicate tasks in process logs," in *Advances in Web and Network Technologies, and Information Management*. Berlin, Germany: Springer, 2007, pp. 396–407.

[8] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. van der Aalst, "Genetic process mining: An experimental evaluation," *Data Mining Knowl. Discovery*, vol. 14, no. 2, pp. 245–304, 2007.

[9] W. van der Aalst, V. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: A two-step approach to balance between underfitting and overfitting," *Softw. Syst. Model.*, vol. 9, no. 1, pp. 87–111, 2010.

[10] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Deriving Petri nets from finite transition systems," *IEEE Trans. Comput.*, vol. 47, no. 8, pp. 859–882, Aug. 1998.

[11] J. Carmona, J. Cortadella, and M. Kishinevsky, "A region-based algorithm for discovering Petri nets from event logs," presented at the 6th Int. Conf. Bus. Process Manage., Milan, Italy, Sep. 2008.

[12] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process mining based on regions of languages," presented at the 5th Int. Int. Conf. Bus. Process Manage., Brisbane, QLD, Australia, Sep. 2007.

[13] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," *Fundam. Inform.*, vol. 94, nos. 3–4, pp. 387–412, 2010.

[14] C. W. Günther and W. M. P. van der Aalst, "Fuzzy mining—Adaptive process simplification based on multi-perspective metrics," presented at the 5th Int. Conf. Bus. Process Manage., Brisbane, QLD, Australia, Sep. 2007.

[15] A. J. M. M. Weijters and J. T. S. Ribeiro, "Flexible heuristics miner (FHM)," in *Proc. IEEE Symp. Comput. Intell. Data Mining (CIDM)*, Paris, France, Apr. 2011, pp. 310–317.

[16] A. J. M. M. Weijters and W. M. P. van der Aalst, "Rediscovering workflow models from event-based data using little thumb," *Integr. Comput.-Aided Eng.*, vol. 10, no. 2, pp. 151–162, 2003.

[17] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs—A constructive approach," in *Proc. Int. Conf. Appl. Theory Petri Nets Concurrency*. Berlin, Germany: Springer, 2013, pp. 311–329.

[18] W. Li, H. Zhu, W. Liu, D. Chen, J. Jiang, and Q. Jin, "An anti-noise process mining algorithm based on minimum spanning tree clustering," *IEEE Access*, vol. 6, pp. 48756–48764, 2018.

[19] O. Kalynychenko, S. Chalyi, Y. Bodyanskiy, V. Golian, and N. Golian, "Implementation of search mechanism for implicit dependences in process mining," in *Proc. Int. Conf. Intell. Data Acquisition Adv. Comput. Syst. (IDAACS)*, Berlin, Germany, Sep. 2013, pp. 138–142.

[20] R. Sarno, P. L. I. Sari, D. Sunaryono, B. Amaliah, and I. Mukhlash, "Mining decision to discover the relation of rules among decision points in a non-free choice construct," in *Proc. Int. Conf. Inf., Commun. Technol. Syst. (ICTS)*, Surabaya, Indonesia, Sep. 2014, pp. 53–58.

[21] W. Zheng, Y. Du, L. Qi, and L. Wang, "A method for repairing process models containing a choice with concurrency structure by using logic Petri nets," *IEEE Access*, vol. 7, pp. 13106–13120, 2019.

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

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

[24] Z. Y. He, Y. Du, L. Wang, L. Qi, and H. C. Sun, "An alpha-FL algorithm for discovering free loop structures from incomplete event logs," *IEEE Access*, vol. 6, pp. 27885–27901, 2018.

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

[26] Y. X. Teng, Y. Y. Du, L. Qi, and W. J. Luan, "A logic Petri net-based method for repairing process models with concurrent blocks," *IEEE Access*, vol. 7, pp. 8266–8282, 2018.

[27] J. Lekić and D. Milićev, "Discovering block–structured parallel process models from causally complete event logs," *J. Elect. Eng.*, vol. 67, no. 2, pp. 111–123, 2016.

[28] Y. Y. Wang and Y. Y. Du, "Conformance checking based on extended footprint matrix," *J. Shandong Univ. Sci. Technol., Nature Sci.*, vol. 37, no. 2, pp. 9–15, 2018.

[29] 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.

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

[31] S. Wang, C. Wang, and M. Zhou, "Controllability conditions of resultant siphons in a class of Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 42, no. 5, pp. 1206–1215, Sep. 2012.

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

[33] S. Wang, D. You, and M. Zhou, "A necessary and sufficient condition for a resource subset to generate a strict minimal siphon in S 4PR," *IEEE Trans. Autom. Control*, vol. 62, no. 8, pp. 4173–4179, Aug. 2017.

[34] S. Zhang, N. Wu, Z. Li, T. Qu, and C. Li, "Petri net-based approach to short-term scheduling of crude oil operations with less tank requirement," *Inf. Sci.*, vol. 417, pp. 247–261, Nov. 2017.

[35] Q. Zhu, M. Zhou, Y. Qiao, and N. Wu, "Petri net modeling and scheduling of a close-down process for time-constrained single-arm cluster tools," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 3, pp. 389–400, Mar. 2018.

[36] D. Xiang, G. Liu, C. Yan, and C. Jiang, "Detecting data-flow errors based on Petri nets with data operations," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 1, pp. 251–260, Jan. 2018.

[37] S. Wang, D. You, and C. Seatzu, "A novel approach for constraint transformation in Petri nets with uncontrollable transitions," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 8, pp. 1403–1410, Aug. 2018.

[38] D. M. Xiang, G. J. Liu, C. G. Yan, and C. J. Jiang, "A guard-driven analysis approach of workflow net with data," *IEEE Trans. Services Comput.*, to be published. doi: 10.1109/TSC.2019.2899086.

[39] Q. Hu, Y. Y. Du, and S. X. Yu, "Service net algebra based on logic Petri nets," *Inf. Sci.*, vol. 268, pp. 271–289, Jun. 2014.

[40] C. Liu, H. Duan, Q. Zeng, M. Zhou, F. Lu, and J. Cheng, "Towards comprehensive support for privacy preservation cross-organization business process mining," *IEEE Trans. Service Comput.*, to be published. doi: 10.1109/TSC.2016.2617331.

[41] L. Qi, M. C. Zhou, and W. J. Luan, "A two-level traffic light control strategy for preventing incident-based urban traffic congestion," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 13–24, Jan. 2016.

[42] L. Qi, M. Zhou, and W. Luan, "Impact of driving behavior on traffic delay at a congested signalized intersection," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 7, pp. 1882–1893, Jul. 2017.

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

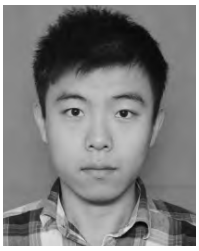
[44] B. F. van Dongen, A. K. A. de Medeiros, and H. M. W. Verbeek, "The ProM framework: A new era in process mining tool support," in *Proc. Int. Conf. Appl. Theory Petri Nets*, Miami, FL, USA, 2005, pp. 444–454.

[45] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. M. P. van der Aalst, "Measuring precision of modeled behavior," *Inf. Syst. e-Bus. Manage.*, vol. 13, no. 1, pp. 37–67, Feb. 2015.

HUIMING SUN received the B.S. degree from the Shandong University of Science and Technology, Qingdao, China, in 2017, 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 in over ten projects supported by the National Natural Science Foundation, the National Key Basic Research Developing Program, and other important and key projects at provincial levels. He has published over 140 papers in domestic and international academic publications, and they are embodied over 80 times by SCI and EI and cited over 270 times by others. His research interests include formal engineering, Petri nets, real-time systems, Web services, and workflows. He is a member of the Professional Committee of Petri nets of the China Computer Federation.

LIANG QI (S'16–M'18) received the B.S. degree in information and computing science and the M.S. degree in computer software and theory from the Shandong University of Science and Technology, Qingdao, China, in 2009 and 2012, respectively, and the Ph.D. degree in computer software and theory from Tongji University, Shanghai, China, in 2017. From 2015 to 2017, he was a Visiting Student with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA. He is currently with the Shandong University of Science and Technology. He has authored over 35 papers in journals and conference proceedings, including the IEEE TRANSACTIONS ON SYSTEM, MAN AND CYBERNETICS: SYSTEMS, the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, the IEEE/CAA JOURNAL OF AUTOMATICA SINICA, the IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS, the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, and the IEEE TRANSACTIONS ON CYBERNETICS. His interests include Petri nets, machine learning, intelligent transportation systems, and optimization. He received the Best Student Paper Award-Finalist at the 15th IEEE International Conference on Networking, Sensing and Control (ICNSC'2018).



ZHAOYANG HE 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.