Refactor Business Process Models for Efficiency Improvement

Fei Dai[1,4], Miao Liu[2], Qi Mo[2,4(✉)], Bi Huang[1], and Tong Li[3,4]

[1] School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming, China
[2] School of Software, Yunnan University, Kunming, China
moqiyueyang@163.com
[3] School of Big Data, Yunnan Agricultural University, Kunming, China
[4] Key Laboratory for Software Engineering of Yunnan Province, Kunming, China

Abstract. Since business processes describe the core value chain of enterprises, thousands of business processes are modeled in business process models. A problem is how to improve the efficiency of these models. In this paper, we propose an approach to refactor these models for efficiency improvement. More specifically, we first identify false sequence relations that affect model efficiency based on the sequence relation matrix and the dependency relation matrix. Second, we refactor a business process model by constructing and transforming a dependency graph without altering its output result. After refactoring, the concurrent execution of business tasks in the original models can be maximized such that its efficiency can be improved. Experimental results show the effectiveness of our approach.

Keywords: Business process model · Efficiency improvement · Refactor · Petri net

1 Introduction

Since business processes describe the core value chain of enterprises, many business managers are attracted to build Process-Aware Information System (PAIS). With the board use of PAIS, thousands of business process models [1] are modeled for a variety of purposes, e.g. process analysis and process enactment. Since modeling business process is error-prone [2], the quality of models is difficult to be guaranteed [3]. When modeling a business process model, a question that arises here is that, can we improve its efficiency? For example, two tasks that can be executed concurrently have been modeled as being executed sequentially. Consequentially, the enactment of this process model is inefficient.

Although there is much work on model soundness analysis [4], i.e. proper termination and no dead tasks, this technique cannot be used to improve the model's efficiency. Thus, we focus on refactoring business process models for efficiency improvement.

In our approach, we first identify false sequence relations that affect model efficiency. More specifically, given a business process model, we check whether two business tasks with a sequence relation has a data dependency relation. If the two tasks has no data

X. Zhang et al. (Eds.): CloudComp 2019/SmartGift 2019, LNICST 322, pp. 454–467, 2020.
https://doi.org/10.1007/978-3-030-48513-9_37

dependency relation, the sequence relation among them is false. Second, we refactor a business process model using a dependency graph. After refactoring, tasks in the original business process models can be maximized concurrent execution, so their efficiency can be improved.

In this work, we make the following main contributions:

- We identify false sequence relations based on the sequence relation matrix and the dependency relation matrix.
- We refactor business process models for efficiency improvement by maximizing the task's concurrent execution.

We organize the rest of this article as follows. Section 2 introduces our motivation and related work. Section 3 presents the definitions used throughout this paper. Section 4 presents a refactoring process for efficiency improvement. Section 5 evaluates the effectiveness of our approach. Section 6 concludes our work and points out one future direction.

2 Motivation and Related Work

2.1 Motivation

Figure 1 shows an inefficient online shopping purchase process in [5]. The circles denote places which can have tokens and the rectangles denote tasks. First, goods are bought via the Internet (task *A*). As a result of task *A*, the outputs are the buyer's address (variable *x*) and the money to be paid. Then, goods are shipped (task *B*). The input of task *B* is the buyer's address (variable *x*) and the output is the goods declaration (variable *z*). Once task *C* is executed (i.e., the goods are received), the input of task *C* is the goods declaration (variable *z*) and the output is the sign for receipt (variable *s*). Then, the bill is sent to the buyer (task *D*). The input of task D is the buyer pays the amount (variable *y*) and the output is the payment request (variable *p*). After task *E* is executed (i.e., the bill is paid by the buyer), the input of task *E is* the payment request (variable *p*) and the output is the completed payment (variable *q*). Finally, this transaction is archived by the seller (task *F*), the input of task *F* is the buyer's signature (variable *s*) and the completed payment (variable *q*).

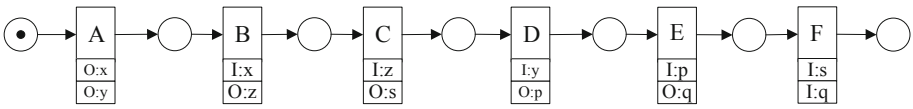


Fig. 1. An inefficient business process model

All the tasks must be executed sequentially in Fig. 1. Since task *D* can be executed after task *C* has been completed, there is a sequence relation between them. However, from a data perspective, the sequence relation is false. Since there is no intersection

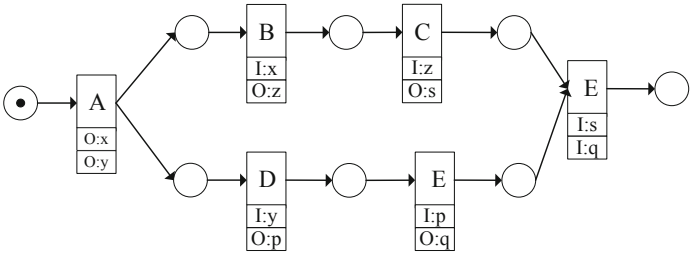


Fig. 2. A refactored business process model with more efficiency

between tasks *C* and *D* (i.e., tasks *C* and *D* have no data dependency), the two tasks can be executed concurrently.

Based on the analysis, the original model in Fig. 1 is inefficient. After refactoring, Fig. 2 shows the refactored model. Intuitively, the refactored model takes less time to execute, i.e., the refactored model is more efficient than the original model. So this refactoring technique can change the false sequence relation between two tasks in the original process model into a concurrence relation, which maximizes the concurrent execution of the two tasks.

2.2 Related Work

To improve the quality of activity tags, the authors proposed an approach based on corpus's second-order similarity in [6] to automatically annotate activity tags in the business process model. Further, the authors investigated how the activity tag style affects models'comprehensibility in [7]. The authors proposed an automatic refactoring technique for converting the activity tag' style in [8]. These works mentioned above mainly focuses on the quality of the model from the view of activity tags. However, our work focuses on the model's efficiency improvement.

To improve the comprehensibility and maintainability of models, the authors summed up eight process model smells to help process designers identify the process model with low quality in [9]. The authors proposed a refactoring technique based on the measure of activity tag consistency and process overlap, which can automatically identify the opportunity to apply four refactoring operations in [10]. The authors focused on selecting the appropriate business process model operation sets by evaluating the quality of comprehensibility and modifiability in [11]. These works mentioned above mainly focuses on focuses on the comprehensibility and maintainability of the model. However, they also cannot be used to improve the efficiency of models.

The only work on refactoring models for efficiency improvement is in [5]. The authors proposed an approach to refactor business process models for efficiency improvement using process mining technology. Our work is different with Jin's work in that our approach can ensure that the refactored models are structured.

3 Preliminaries

Petri nets are often used to model business process models, due to its formalization foundation and analysis technique.

Definition 1 **(*Petri net*):** A Petri net is a triple $N = (P, T, F)$ where:

- P is a set of places;
- T is a set of transitions;
- $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation.

State M is the marking of N and can be defined as: $M{:}P \rightarrow \{0,1,2,3, \ldots\}$. M_0 often refers to the initial state of N.

Denote $X = (P \cup T)$, for a node $u \in X$, ${}^{\bullet}u = \{v \in X \mid (v, u) \in F\}$ is called the preset of u; $u^{\bullet} = \{v \in X \mid (u, v) \in F\}$ is called the postset of u.

Definition 2 **(*WF-net*):** A Petri net $N = (P, T, F)$ is a WF-net(workflow net), iff:

- $i \in P$ is a source place such that ${}^{\bullet}i = \phi$;
- $o \in P$ is a sink place such that $o^{\bullet} = \phi$; and
- $x \in P \cup T$ is on a path from i to o.

WF-net is a subclass of Petri net. It was first proposed in [12] and was used to model business processes.

Definition 3 **(*DWF-net*) [5]:** A $DWN = (P, T, F, D, Input, Output)$ is a DWF-net (workflow net with data), where:

- $N = (P, T, F)$ is a WF-net;
- D is the set of data consumed and provided by WF-net;
- *Input:* $T \rightarrow D$ denotes the set of data consumed by tasks,
- *Output:* $T \rightarrow D$ denotes the set of data provided by tasks.

If a DWF-net is unstructured, the work in [13] and [14] discussed how to convert unstructured DWF-nets into structured DWF-nets in detail. Thus we suppose that each DWF-net is structured.

Definition 4 **(*Direct sequence relation*):** Let $DWN = (P, T, F, D, Input, Output)$ be a workflow net with data, $t_1, t_2 \in T$. We say t_1 and t_2 have a direct sequence relation, denoted as $t_1 > t_2$, if there exists a place $p \in P$, such that $t_1 \in {}^{\bullet}p \wedge t_2 \in p^{\bullet} \wedge |{}^{\bullet}p| = 1 \wedge |p^{\bullet}| = 1$.

Definition 5 **(*Transitive sequence relation*):** Let $DWN = (P, T, F, D, Input, Output)$ be a workflow net with data, $t_1, t_2, t_3 \in T$. We say t_1 and t_3 have a transitive sequence relation, denoted as $t_1 \gg t_3$, if $t_1 > t_2 \wedge t_2 > t_3$.

In this paper, direct sequence relations and transitive sequence relations are called sequence relations.

Definition 6 ***(Direct data dependency relation)*****:** Let $DWN = (P, T, F, D, Input, Output)$ be a workflow net with data, $t_1, t_2 \in T$, t_1 is executed before t_2, there is a direct data dependency relation denoted as $t_1 \delta^d t_2$, if t_1 and t_2 satisfy one of the following conditions:

- $Output(t_1) \cap Input(t_2) \subseteq D$. That is, there is a true-dependency relation between t_1 and t_2.
- $Output(t_2) \cap Input(t_1) \subseteq D$. That is, there is an anti-dependency relation between t_1 and t_2 t_1 and t_2.
- $Output(t_1) \cap Output(t_2) \subseteq D$. That is, there is an output-dependency relation between t_1 and t_2 t_1 and t_2.

The relation of δ^d can be transitive.

Definition 7 **(*Transitive data dependency relation*):** Let $DWN = (P, T, F, D, Input, Output)$ be a workflow net with data, $t_1, t_2, t_3 \in T$, there is a transitive data dependency relation denoted as $t_1 \delta^* t_3$ if $t_1 \delta^d t_2 \wedge t_2 \delta^d t_3$.

Transitive data dependency relations are also transitive. In this paper, direct data dependency relations and transitive data dependency relations are called data dependency relations denoted as δ.

Definition 8 **(Control dependency relation):** Let $DWN = (P, T, F, D, Input, Output)$ be a workflow net with data, $t_1, t_2 \in T$, t_1 is executed before t_2, there is a control dependency relation between t_1 and t_2, denote as $t_1 \delta^c t_2$, iff $t_1 \in {}^\bullet p \ \wedge t_2 \in p^\bullet \wedge \neg(|{}^\bullet p| = 1 \wedge |p^\bullet| = 1)$.

4 Our Proposed Approach

Figure 3 shows an overview of our approach, which consists of five steps.

- Step 1: construct a sequence relation matrix from the given business process model.
- Step 2: construct a dependency relation matrix from the given business process model.
- Step 3: identify false sequence relations based on the sequence relation matrix and the dependency relation matrix.
- Step 4: construct a dependency graph.
- Step 5: transform a dependency graph into a DWF-net.

4.1 Construct a Sequence Relation Matrix

According to the α mining algorithm [15], there are three types of task relations, namely sequence relation, selection relation, and concurrency relation. Since only the sequence relation leads to model inefficiency, we try to construct a sequence relation matrix that records the direct sequence relation or the transitive sequence relation between two tasks in a business process model using Algorithm 1.

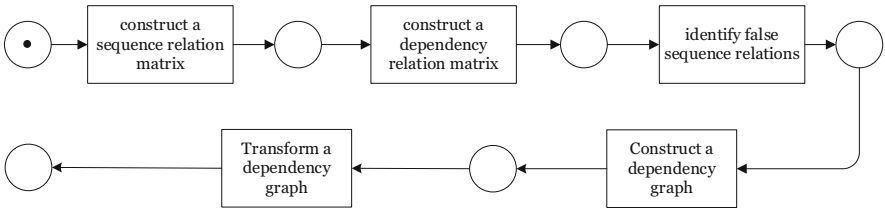


Fig. 3. Overview of our approach

According to definition 4, we obtain all direct sequence relations of $>$ directly by traversing all the places in a business process, which be found in lines 1–9 of Algorithm 1.

According to Definition 5, after direct sequence relations are obtained, the transitive sequence relations of $\gg$ can be computed using the lines 10–18 of Algorithm 1.

Let m and n be separately the number of places and the number of transitions. Algorithm 1's worst time complexity is $O(m{*}n^2 + n^3)$.

Algorithm 1: construct a sequence relation matrix

Input: *DWN*=(*P*, *T*, *F, D, Input, Output*)
Output: a sequence relation matrix $SM_{|T|*|T|}$
1. **for** $p \in P$ **do**
2. **if** $(|{}^\bullet p|==1) \wedge (|p^\bullet|==1)$ **then**
3. **for** $t_1 \in {}^\bullet p$ **do**
4. **for** $t_2 \in p^\bullet$ **do**
5. set $t_1 > t_2$ in *SM*;
6. **end for**
7. **end for**
8. **end if**
9. **end for**
10. **for** $t_1 \in T$ **do**
11. **for** $t_2 \in T$ **do**
12. **for** $t_3 \in T$
13. **if**$((t_1>t_2 \wedge\ t_2>t_3\ \wedge t_1! \gg t_3)\ \vee\ (t_1 \gg t_2 \wedge\ t_2>t_3\ \wedge t_1! \gg t_3) \vee (t_1>t_2 \wedge\ t_2 \gg t_3\ \wedge t_1! \gg t_3)$
$\vee (t_1 \gg t_2 \wedge t_2 \gg t_3\ \wedge t_1! \gg t_3)$ **then**
14. set $t_1 \gg t_3$ in *SM*;
15. **end if**
16. **end for**
17. **end for**
18. **end for**
19. **return** *SM*;

For the model in Fig. 1, we can construct the sequence relation matrix as follows.

$$\begin{bmatrix} & A & B & C & D & E & F \\ A & & > & \gg & \gg & \gg & \gg \\ B & & & > & \gg & \gg & \gg \\ C & & & & > & \gg & \gg \\ D & & & & & > & \gg \\ E & & & & & & > \\ F & & & & & & \end{bmatrix}$$

4.2 Construct a Dependency Relation Matrix

For two transitions having a sequence relation, the data dependency relation between them is either δ^{d}, δ^{*}, δ^{c} or null. In this step, we construct a dependency relation matrix using Algorithm 2. The dependency relation matrix records the direct data dependency relation or the transitive data dependency relation between two tasks in a business process model. Besides, the dependency relation matrix records the control dependency relation between two tasks in a business process model.

According to Definition 6, we can obtain all direct data dependency relations of δ^{d} directly by traversing all the transitions in a business process, which be found in lines 1–7 of Algorithm 2.

According to Definition 7, after direct data dependency relations of δ^{d} is obtained, the transitive data dependency relations of δ^{*} can be computed using the lines 8–16 of Algorithm 2 (lines 8–16).

According to Definition 8, we can obtain all control dependency relations of δ^{c} directly by traversing all the places in a business process, which be found in lines 17–25 of Algorithm 2(lines 17–25).

Let m and n be separately the number of places and the number of transitions. Algorithm 2's worst time complexity is $O(m{*}n^2 + n^3)$.

Algorithm 2: construct a dependency relation matrix

Input: DWN=(P, T, F, D, *Input*, *Output*)
Output: a dependency relation matrix $DM_{|T|*|T|}$

for $t_1 \in T$ **do**
 for $t_2 \in T \wedge (t_1 > t_2 \vee t_1 \gg t_2)$ **do**
 if ($(Output(t_2) \cap Input(t_1) \neq \phi) \vee (Output(t_1) \cap Input(t_2) \neq \phi) \vee (Output(t_1) \cap Output(t_2) \neq \phi)$) **then**
 set $t_1 \delta^d t_2$ in DM;
 end if
 end for
end for
for $t_1 \in T$ **do**
 for $t_2 \in T$ **do**
 for $t_3 \in T$
 if($(t_1 \delta^d t_2 \wedge t_2 \delta^d t_3 \wedge t_1 ! \delta^* t_3) \vee (t_1 \delta^d t_2 \wedge t_2 \delta^* t_3 \wedge t_1 ! \delta^* t_3) \vee (t_1 \delta^* t_2 \wedge t_2 \delta^* t_3 \wedge t_1 ! \delta^* t_3) \vee (t_1 \delta^* t_2 \wedge t_2 \delta^d t_3 \wedge t_1 ! \delta^* t_3)$) **then**
 set $t_1 \delta^* t_3$ in DM;
 end if
 end for
 end for
end for
for $p \in P$ **do**
 if $((|^\bullet p| == 1) \wedge (|p^\bullet| > 1)) \vee ((|^\bullet p| > 1) \wedge (|p^\bullet| == 1))$ **then**
 for $t_1 \in {}^\bullet p$ **do**
 for $t_2 \in p^\bullet$ **do**
 set $t_1 \delta^* t_2$ in DM;
 end for
 end for
 end if
end for
return DM;

For the model in Fig. 1, we can construct the dependency relation matrix as follows.

$$\begin{bmatrix} & A & B & C & D & E & F \\ A & & \delta^d & \delta^* & \delta^d & \delta^* & \delta^* \\ B & & & \delta^d & & & \delta^* \\ C & & & & & & \delta^d \\ D & & & & & \delta^d & \delta^* \\ E & & & & & & \delta^d \\ F & & & & & & \end{bmatrix}$$

4.3 Identify False Sequence Relations

We identify false sequence relations using Algorithm 3. If the two tasks have a sequence relation in a sequence relation matrix but have no data dependency relation in the corresponding dependency relation matrix, the sequence relation is false. The false sequence relation can be a false adjacent sequence relation (for two adjacent transitions on the same path) or a false transitive sequence relation (for two not adjacent transitions on the same path).

1) a false adjacent sequence relation: If two tasks in a business process model have an adjacent sequence relation in its sequence relation matrix but no direct data dependency relation in its corresponding data dependency relation matrix, the adjacent sequence relation is false.
2) a false transitive sequence relation: If two tasks in a business process model have a transitive sequence relation in its sequence relation matrix but no transitive data dependency relation in its corresponding data dependency relation matrix, the transitive sequence relation is false.

Let n be the number of transitions in a business process model. The worst time complexity of Algorithm 3 is $O(n^2)$.

Algorithm 3: identify false sequence relations

Input: DWN=(P, T, F, D, *Input, Output*), the related $SM_{|T|*|T|}$, and the $DM_{|T|*|T|}$
Output: the set of false sequence relations FSR
1 **for** $t_1 \in T$ **do**
2 **for** $t_2 \in T$ **do**
3 **if** $((t_1,t_2) == >$ in $SRM) \wedge ((t_1,t_2) == \phi$ in $DM)$ **then**
4 $FSR = FSR \cup \{(t_1,t_2)\}$;
5 **end if**
6 **if** $((t_1,t_2) == \gg$ in $SRM) \wedge ((t_1,t_2) == \phi$ in $DM)$ **then**
7 $FSR = FSR \cup \{(t_1,t_2)\}$;
8 **end if**
9 **end for**
10 **end for**
11 **return** FSR;

For the model in Fig. 1, we can see that the direct sequence relation between the two tasks C and D is false and that the transitive sequence relations between tasks B and D, C and E, B and E are false.

4.4 Construct a Dependency Graph

We construct a dependency graph using Algorithm 4, which can be used to describe data dependency relations between two tasks intuitively. In the dependency graph, the two tasks with connected arcs must be executed sequentially while the two tasks without arcs can be executed concurrently. In general, the circles denote tasks, the directed edges denote arcs, and the label of each directed edge denotes the type of dependency relations.

***Definition* 9 (Dependency graph)**: A dependency graph is a triple $DG = (T, A, R)$, where:

- T is the set of tasks ($T \neq \phi$);
- $A \subseteq T \times T$ is the set of arcs;
- R: $A \rightarrow \{\delta^{d}, \delta^{*}, \delta^{c}\}$is a labeling function that assigns to each arc x of A a label $R(x)$.

According to Definition 9, we can traverse all the transitions in a business process to obtain a dependency graph directly. The algorithm of constructing a dependency graph can be found in Algorithm 4.

Let n be the number of transitions in a business process model. The worst time complexity of Algorithm 4 is $O(n^2)$.

Algorithm 4: construct a dependency graph

Input: DWN=(P, T, F, D, *Input, Output*), the related dependency matrix $DM_{|T|*|T}$
Output: the dependency graph DG
1 $DG.T$=$DN.T$
2 **for** $t_1 \in T$ **do**
3 **for** $t_2 \in T$ **do**
4 **if** ((t_1,t_2)==δ^d in DM) **then**
5 set A= $A \cup \{(t_1,t_2)\}$; R=$R \cup \{((t_1,t_2), \delta^d)\}$
6 **end if**
7 **if** ((t_1,t_2)==δ^* in DM) **then**
8 set A= $A \cup \{(t_1,t_2)\}$; R=$R \cup \{((t_1,t_2), \delta^*)\}$
9 **end if**
10 **if** ((t_1,t_2)==$\delta^{\mathcal{E}}$ in DM) **then**
11 set A= $A \cup \{(t_1,t_2)\}$; R=$R \cup \{((t_1,t_2), \delta^{\mathcal{E}})\}$
12 **end if**
13 **end for**
14 **end for**
15 **return** DG

For the model in Fig. 1, we construct its dependency graph, which is shown in Fig. 4.

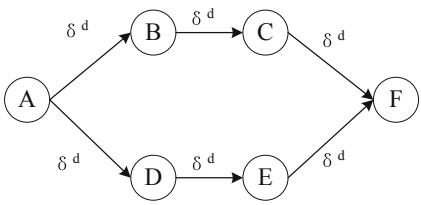


Fig. 4. A dependency graph for the process model in Fig. 1

4.5 Transform a Dependency Graph

Transform a dependency graph means to transform a dependency graph into a DWF-net according to the transforming rules using Algorithm 5. There are four steps:

1) transform node: each node in the dependency graph is transformed into a transition in the Petri net (rule 1);
2) transform sequence structure: each graph segment as shown in Table 1 (rule 2) in the dependency graph is transformed into a sequence structure;
3) transform selection structure:each graph segment as shown in Table 1 (rule 3 and rule 4) in the dependency graph is transformed into a selection structure;
4) add extra places: add the start place and the end place.

Algorithm 5: transform a dependency graph

Input: $DWN=(P, T, F, D, Input, Output)$ and its related simplified dependency graph $SDG = (T, A, R)$

Output: a structured workflow net with data $DN_r=(P_r, T_r, F_r, M_r, D_r, Input_r, Output_r)$

//transform nodes

1. $T_r=T$; $Input_r= Input$; $D_r=D$;$O utput_r=Output$;

// transform sequence structures

2. **for** $(t_i, t_j)\in A$ in SDG **do**

3. $P_r= P_r\cup p_{ij}$; $F_r=F_r\cup\{(t_i, p_{ij})\}\cup\{(p_{ij}, t_j)\}$;

4. **end for**

//) transform selection structures

5. **for** $((t_i, t_j), \delta), (t_i, t_k), \delta), \ldots, (t_i, t_s), \delta))\in R$ in SDG **do**

6. $P_r=P_r-\{p_{ik}\}-\ldots.-\{p_{is}\}$; $F_r=F_r-\{(t_i, p_{ik})\}-\ldots-\{(t_i, p_{is})\}-\{(p_{ik}, t_k)\}-\ldots-\{(p_{is},t_s)\}\cup(p_{ij}, t_k)\cup\ldots\cup(p_{ij}, t_s)$;

7. **end for**

8. **for** $((t_i, t_j), \delta), (t_k, t_j), \delta), \ldots, (t_s, t_j), \delta))\in R$ in SDG **do**

9. $P_r=P_r-\{p_{kj}\}-\ldots.-\{p_{sj}\}$; $F_r=F_r-\{(t_k, p_{kj})\}-\ldots-\{(t_s, p_{sj})\}-\{(p_{kj}, t_j)\}-\ldots-\{(p_{sj}, t_j)\}\cup(t_k, p_{ij})\cup\ldots\cup(t_s,p_{ij})$;

10. **end for**

//add the start place and the end place

11. **for** $t_i\in T_r$ **do**

12. **if** ${}^\bullet t_i=\phi$ **then**

13. $Pr=Pr\cup\{p_{satrti}\}$; $Fr=Fr\cup\{(p_{satrti},t_i)\}$;

14. **end if**

15. **if** $t_i^\bullet=\phi$ **then**

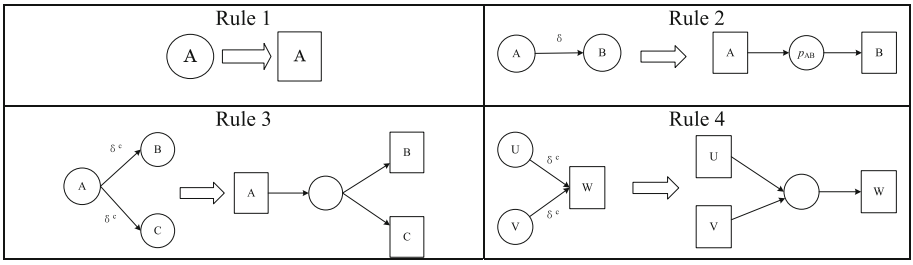
16. $Pr=Pr\cup\{p_{endi}\}$; $Fr=Fr\cup\{(t_i, p_{endi})\}$;

17. **end if**

18. **end for**

19. **return** DN_r

Table 1. Transformation rules



We can transform the dependency graph in Fig. 4 into a DWF-net in Fig. 2 using the above transformation rules.

5 Experiments

We implemented a prototype tool based on PIPE (The Platform Independent Petri Net Editor) [16]. In Fig. 5, the example given in this paper is on the left and the refactored model is on the right.

200 different DWF-nets are generated randomly. All experiments were conducted on the same computer with Inter (R) i5 2.5 GHz and 8 GB RAM, Windows 10 and JDK 7.

To evaluate the effectiveness of our approach, parallelism degree (PD) as Eq. 1 is used to measure the concurrency according to the work in [5, 17]. If the PD of the refactored model is greater than the PD of the original model, it means that our approach can improve the original business process model's efficiency.

$$\mathrm{PD} = \sum\nolimits_{d_{out}(t)>1} (d_{out}(t) - 1) \tag{1}$$

Where $d_{out}(t)$ represents the output degree of task t; $d_{out}(t) > 1$ indicates that the out degree of task t is greater than one. If $\mathrm{PD} = -1$, no task in the business process model can execute concurrently.

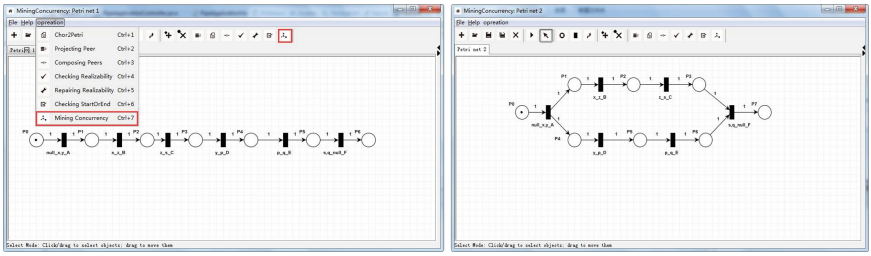


Fig. 5. The screenshot of tool prototype

Table 2 presents some experimental results, including the tasks with $d_{out}(t) > 1$ in the original model ($\mathrm{number_b}$), the parallelism degree in the original model ($\mathrm{PD_b}$), hit before refactoring($\surd$ indicates the original model is structured and $\times$ indicates the original model is not structured), the tasks with $d_{out}(t) > 1$ in the refactored model ($\mathrm{number_a}$), the parallelism degree in the refactored model ($\mathrm{PD_a}$), and hit after refactoring ($\surd$ indicates the refactored model is structured and $\times$ indicates the refactored model is not structured) in each example.

From the experimental results, we can see that: 1) In each example, the PD of the refactored model is greater than the PD of the original model. That is, our approach can improve the original business process model's efficiency. 2) All the business process models are structured before and after refactoring. That is, our approach can ensure that the refactored models are structured.

Table 2. Evaluation results

No.	The original model			The refactored model		
	$Number_b$	PD_b	$Structuredness_b$	$Number_a$	PD_a	$Structuredness_a$
1	0	0	√	1	1	√
2	0	0	√	2	4	√
3	2	2	√	4	5	√
4	0	0	√	1	2	√
5	0	0	√	2	2	√
6	1	2	√	2	3	√
7	0	0	√	2	2	√
8	1	1	√	3	3	√
9	0	0	√	1	1	√
10	1	1	√	1	2	√

6 Conclusions

This paper proposed an approach for refactoring process models for efficiency improvement. First, we identify false sequence relations based on the sequence relation matrix and the dependency relation matrix. Second, we refactor a business process model by constructing and transforming a dependency graph. In a refactored model, the concurrent execution of business tasks in the original models can be maximized. Finally, we present a prototype implementation and evaluate the effectiveness of our approach.

The next step will focus on refactoring business process models with inefficient process fragments substitution. Inefficient process fragments refer to sequence process fragments with false sequence relations. If we can replace these inefficient process fragments in an original process model with efficient process fragments, its efficiency can be improved.

Acknowledgment. This work was supported by NSFC (No. 61702442 and 61862065), and the Basic Research Project in Yunnan Province (2018FB105).

References

1. Leopold, H., Mendling, J., Reijers, H.A., Rosa, M.L.: Simplifying process model abstraction: techniques for generating model names. Inf. Syst. **39**, 134–151 (2014)
2. Herbst, J., Karagiannis, D.: Workflow mining with InWoLvE. Comput. Ind. **53**, 245–264 (2004)
3. Khlif, W., Ben-Abdallah, H.: Integrating semantics and structural information for BPMN model refactoring. In: Proceedings of the International Conference on Computer and Information Science, pp. 656–660. IEEE (2015)

4. Aalst, W.M.P.: Workflow verification: finding control-flow errors using petri-net-based techniques. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_11
5. Jin, T., Wang, J., Yang, Y., Wen, L., Li, K.: Refactor business process models with maximized parallelism. IEEE Trans. Serv. Comput. **9**, 456–468 (2016)
6. Leopold, H., Meilicke, C., Fellmann, M., Pittke, F., Stuckenschmidt, H., Mendling, J.: Towards the automated annotation of process models. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 401–416. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19069-3_25
7. Mendling, J., Reijers, H.A., Recker, J.: Activity labeling in process modeling: empirical insights and recommendations. Inf. Syst. **35**, 467–482 (2010)
8. Leopold, H., Smirnov, S., Mendling, J.: Refactoring of process model activity labels. In: Hopfe, Christina J., Rezgui, Y., Métais, E., Preece, A., Li, H. (eds.) NLDB 2010. LNCS, vol. 6177, pp. 268–276. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13881-2_28
9. Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring large process model repositories. Comput. Ind. **62**, 467–486 (2011)
10. Dijkman, R., Gfeller, B., Küster, J., Völzer, H.: Identifying refactoring opportunities in process model repositories. Inf. Softw. Technol. **53**, 937–948 (2011)
11. Fernández-Ropero, M., Pérez-Castillo, R., Caballero, I., Piattini, M.: Quality-driven business process refactoring. In: Proceedings of the International Conference on Business Information Systems (ICBIS 2012), pp. 960–966 (2012)
12. Van der Aalst, W.M.P.: The application of Petri nets to workflow management. J. Circuits Syst. Comput. **8**, 21–66 (1998)
13. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Inf. Syst. **37**, 518–538 (2010)
14. Polyvyanyy, A., García-Bañuelos, L., Fahland, D., Weske, M.: Maximal structuring of acyclic process models. Comput. J. **57**, 12–35 (2014)
15. Van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
16. Dingle, N.J., Knottenbelt, W.J., Suto, T.: PIPE2: a tool for the performance evaluation of generalised stochastic Petri nets. ACM SIGMETRICS Perform. Eval. Rev. **36**, 34–39 (2009)
17. Mendling, J.: Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89224-3