Detecting Implicit Dependencies Between Tasks from Event Logs

Lijie Wen, Jianmin Wang, and Jiaguang Sun

School of Software, Tsinghua University, 100084, Beijing, China
wenlj00@mails.tsinghua.edu.cn, {jimwang, sunjg}@tsinghua.edu.cn

Abstract. Process mining aims at extracting information from event logs to capture the business process as it is being executed. In spite of many researchers' persistent efforts, there are still some challenging problems to be solved. In this paper, we focus on mining non-free-choice constructs, where the process models are represented in Petri nets. In fact, there are totally two kinds of causal dependencies between tasks, i.e., explicit and implicit ones. Implicit dependency is very hard to mine by current mining approaches. Thus we propose three theorems to detect implicit dependency between tasks and give their proofs. The experimental results show that our approach is powerful enough to mine process models with non-free-choice constructs.

1 Introduction

Nowadays, more and more organizations are introducing workflow technology to manage their business processes. Setting up and maintaining a workflow management system requires process models which prescribe how business processes should be managed. Typically, users are required to provide these models. However, constructing process models from scratch is a difficult, time-consuming and error-prone task that often requires the involvement of experts. Furthermore, there are often discrepancies between the predefined models and those really being executed. Process mining is an alternative way to construct process models. It distills process models from event logs that are recorded by most transactional information systems, such as ERP, CRM, SCM and B2B. Process mining can also be used to analyze and optimize prefined process models.

In many cases, the benefit of process mining depends on the exactness of the mined models [1]. The mined models should preserve all the tasks and the dependencies between them that are present in the logs. Although much research is done in this area, there are still some challenging problems to be solved [2, 3, 4]. This paper focuses on mining non-free-choice constructs. For process models with this kind of construct, the key factor affecting the exactness of the mined models is whether dependencies between tasks can be mined correctly and completely. There are totally two kinds of dependencies between tasks, i.e., explicit and implicit ones, which will be introduced in detail later.

The rest of this paper is organized as follows. Section 2 gives some preliminaries about process mining. Section 3 defines explicit and implicit dependencies between tasks and gives all cases in which implicit dependencies must be detected correctly. Section 4 gives three theorems with proofs for detecting implicit dependencies. Experimental results are given in Section 5. Section 6 introduces related work and Section 7 concludes this paper and gives our future work.

2 Preliminaries

In this section, we give some definitions used throughout this paper. Firstly, we introduce a workflow process modelling language and its relevant concepts. Then we discuss the workflow log in detail and give an example.

2.1 WF-Net

In this paper, WF-net is used as the workflow process modelling language, which was proposed in [5]. WF-nets are a subset of Petri nets, which are well understood. Note that Petri nets provide a graphical but formal language designed for modelling concurrency. Moreover, Petri nets provide all kinds of routings supported by workflow management systems in a natural way.

Fig. 1 gives an example of a workflow process modelled in WF-net. This model has a non-free-choice construct. The transitions (drawn as rectangles) T_1, T_2, $\cdots$, T_5 represent tasks and the places (drawn as circles) P_1, P_2, $\cdots$, P_6 represent causal dependencies. A place can be used as pre-condition and/or post-condition for tasks. The arcs (drawn as directed edges) between transitions and places represent flow relations. In this process model, there is a non-free-choice construct, i.e., the sub-construct composed of P_3, P_4, P_5, T_4 and T_5. For T_4, T_5 and the connected arcs, their common input set is not empty but their input sets are not the same.

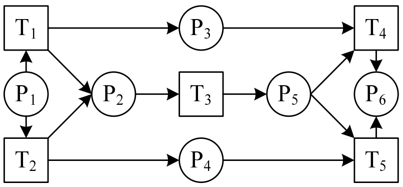

Fig. 1. An example of a workflow process in WF-net

We adopt the formal definitions and properties (such as soundness and safeness) of WF-net and SWF-net from [5, 6]. Some relative definitions (such as implicit place), properties and firing rules about Petri nets are also described there.

For mining purpose, we demand that each task (i.e., transition) has an unique name in one process model. However, each task can appear multiple times in one process instance for the presence of the iterative routings.

2.2 Workflow Log

The goal of process mining is to extract information about processes from transaction logs. We assume that it is possible to record events such that (i) each event refers to a task (i.e., a well-defined step in the process), (ii) each event refers to a case (i.e., a process instance), and (iii) events are totally ordered. Any information system using transactional systems such as ERP, CRM, or workflow management systems will offer this information in some form [6].

Through sorting all the events in a workflow log by their process identifier and complete time, we need only consider that an event has just two attributes, i.e., task name and case identifier. Table 1 gives an example of a workflow log.

Table 1. A workflow log for the process shown in Fig.1

Case id	Task name	Case id	Task name
1	T_1	2	T_2
1	T_3	2	T_3
1	T_4	2	T_5

This log contains information about two cases. The log shows that for case 1, T_1, T_3 and T_4 are executed. For case 2, T_2, T_3 and T_5 are executed. In fact, no matter how many cases there are in the workflow log, there are always only two distinct workflow traces, i.e., $T_1T_3T_4$ and $T_2T_3T_5$. Thus for the process model shown in Fig.1, this workflow log is a minimal and complete one. Here we adopt the definitions of workflow trace and workflow log from [6].

3 Dependency Classification

To distill a process model with non-free-choice constructs from event logs correctly, we must find a way to mine all the dependencies between tasks without mistakes. As research results show, not all dependencies can be mined directly now. Look backwards to the WF-net shown in Fig.1. It is a typical WF-net with a non-free-choice construct. Firstly, there is a free choice between T_1 and T_2. After one of them is chosen to execute, T_3 is executed. Finally, whether T_4 or T_5 is chosen to execute depends on which one of T_1 and T_2 has been executed. There is a non-free-choice between T_4 and T_5. If we apply α algorithm [6] on the log shown in Table 1, P_3 and P_4 as well as their input and output arcs could not be mined. So the mined model is not behavior equivalent with the original model. It can generate two additional kinds of workflow traces, i.e., $T_1T_3T_5$ and $T_2T_3T_4$. Obviously, this result is not what we want.

In fact, there are totally two kinds of dependencies between tasks which must be mined from event logs, i.e., explicit and implicit ones. *Explicit dependency*, which is also called *direct dependency*, reflects direct causal relationships between tasks. *Implicit dependency*, which is also called *indirect dependency*, reflects indirect causal relationships between tasks. As Fig.1 shows, P_2 together with its surrounding arcs reflects explicit dependencies between T_1 and T_3 as well as T_2 and T_3. While P_3 together with its surrounding arcs reflects implicit dependencies between T_1 and T_4. If there are only explicit dependencies between tasks in a process model with non-free-choice constructs, most process mining algorithms, such as the α algorithm etc., can mine it correctly. Otherwise, no algorithm can mine it successfully.

Now we investigate what characteristics a process model with implicit dependencies may have. Assume that there is an implicit dependency between A and B. Once A is executed, there must be some other tasks before B to be executed. After that, B is to be executed. There is no chance that B can directly follow A in some workflow trace. So the implicit dependency between A and B has

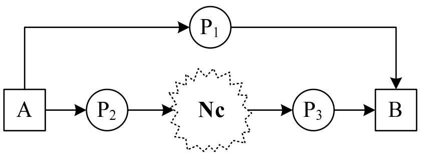

Fig. 2. Characteristics of a process model with implicit dependency

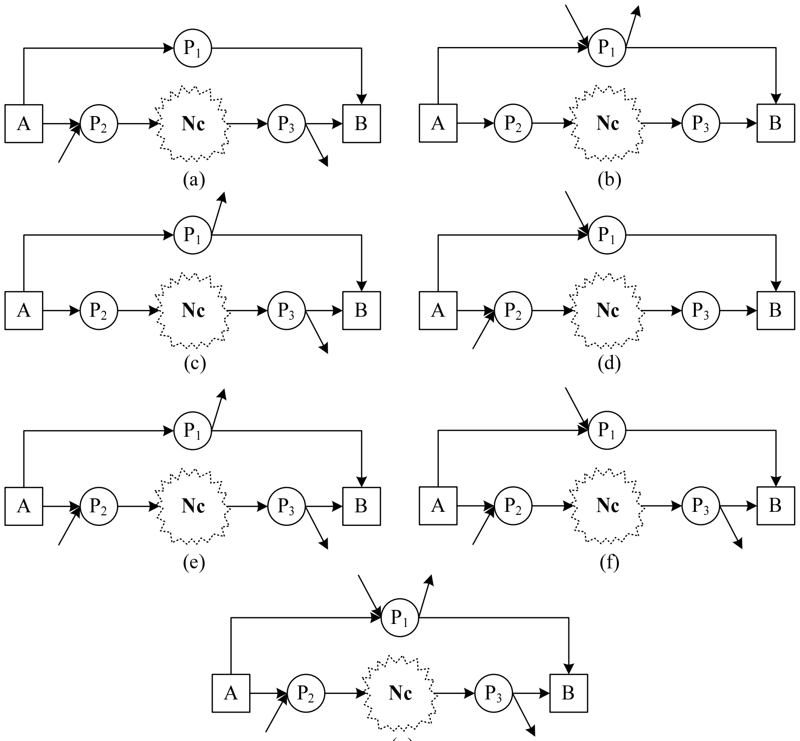

Fig. 3. Sound sub-WF-nets with implicit dependency

no chance to be detected. This typical characteristics of a process model with implicit dependency is shown in Fig.2.

The subnet N_c contains at least one task. It takes tokens from P_2 and puts tokens into P_3. In a general case, there may be more complicated relationships between N_c and the rest of the process model. However, we only consider the simplest case and other cases can be converted to this case easily. Therefore we need not consider the cases where some tasks outside of N_c take P_2 as their input place or P_3 as their output place. If there are no other tasks connected to P_1, P_2 and P_3, P_1 becomes an implicit place. Implicit place does not do any help for the behaviors of a process model. Any sound mining algorithm should avoid constructing implicit places. By enumerating and analyzing all the cases inherited from the simplest case, all of which refer to the input and output tasks of P_1, the input tasks of P_2 and the ouput tasks of P_3, totally seven kinds of sub-WF-nets are proven to be possibly sound. All of these seven sub-WF-nets are shown in Fig.3. Lack of spaces forbids the detail proofs of the above conclusions.

For sub-WF-nets (a) shown in Fig.3, place P_1 and its surrounding arcs will not be mined. For (b) and (g), place P_1 may be replaced by two or more places. For (c) and (e), the arc (P_1,B) will be omitted. For (d) and (f), the arc (A,P_1) will be omitted. Altogether, there are three distinct cases, i.e., (a), (b) and (g), and (c) to (f). All the relative theorems and their proofs will be given in the next section.

4 Detecting Implicit Dependencies

From the above sections, it is obvious that the detection of implicit dependencies is the most important factor for mining process models with non-free-choice constructs correctly. In this section, we will introduce all the three theorems and their corresponding proofs in detail. There exists a one-to-one relationship between the three theorems and the above three cases of implicit dependencies.

To detect explicit dependencies between tasks, we adopt the α^+ algorithm [7]. Some definitions, such as $>_w$, $\rightarrow_w$, $\#_w$, $\|_w$, etc., are also borrowed from [7] with some modification. Based on these basic ordering relations, we provide some additional new definitions for advanced ordering relations.

Definition 1 (Ordering relations). *Let W be a one-length-loop-free workflow log over T. Let a,b∈T:*

- *$a \triangle_w b$ if and only if there is a trace $\sigma = t_1t_2t_3 \ldots t_n$ and $i \in \{1, \ldots, n-2\}$ such that $\sigma \in W$ and $t_i = t_{i+2} = a$ and $t_{i+1} = b$,*
- *$a >_w b$ if and only if there is a trace $\sigma = t_1t_2t_3 \ldots t_n$ and $i \in \{1, \ldots, n-1\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$,*
- *$a \rightarrow_w b$ if and only if $a >_w b$ and $b \not>_w a$ or $a \triangle_w b$ or $b \triangle_w a$,*
- *$a \#_w b$ if and only if $a \not>_w b$ and $b \not>_w a$,*
- *$a \|_w b$ if and only if $a >_w b$ and $b >_w a$ and $\neg(a \triangle_w b$ or $b \triangle_w a)$,*
- *$a \lhd_w b$ if and only if $a \#_w b$ and there is a task c such that $c \in T$ and $c \rightarrow_w a$ and $c \rightarrow_w b$,*

– *$a \rhd_w b$ if and only if $a \#_w b$ and there is a task c such that $c \in T$ and $a \rightarrow_w c$ and $b \rightarrow_w c$,*
– *$a \gg_w b$ if and only if $a \not>_w b$ and there is a trace $\sigma = t_1 t_2 t_3 \ldots t_n$ and $i, j \in \{1, \ldots, n\}$ such that $\sigma \in W$ and $i < j$ and $t_i = a$ and $t_j = b$ and for any $k \in \{i+1, \ldots, j-1\}$ satisfying $t_k \neq a$ and $t_k \neq b$ and $\neg(t_k \lhd_w a$ or $t_k \rhd_w a)$,*
– *$a \succ_w b$ if and only if $a \rightarrow_w b$ or $a \gg_w b$, and*
– *$a \mapsto_w b$ if and only if b is implicitly dependent on a, or there is an implicit dependency from b to a.*

Definitons of $\triangle_w$, $>_w$ and $\#_w$ are the same as those defined in [7]. Definitions of $\rightarrow_w$ and $\|_w$ are a little different. Given a complete workflow log of a sound SWF-net [6, 7] and two tasks a and b, $a \triangle_w b$ and $b \triangle_w a$ must both come into existence. But for a one-length-loop-free workflow log of a sound WF-net, it is not always true. Now we will turn to the last five new definitions. Relation $\lhd_w$ corresponds to OR-Split while relation $\rhd_w$ corresponds to OR-Join. Relation $\gg_w$ represents that one task can only be indirectly followed by another task. Relation $\succ_w$ represents that one task can be followed by another task directly or indirectly. Relation $\mapsto_w$ represents implicit dependency between tasks. Consider the workflow log shown in Table 1, it can be represented as string sets, i.e., $\{T_1T_3T_4,\ T_2T_3T_5\}$. From this log, the following advanced ordering relations between tasks can be detected: $T_1 \rhd_w T_2$, $T_4 \lhd_w T_5$, $T_1 \gg_w T_4$, $T_2 \gg_w T_5$, $T_1 \mapsto_w T4$ and $T_2 \mapsto_w T_5$. For any two tasks a and b, $a \rightarrow_w b$ and $a \mapsto_w b$ cannot be true at the same time. Here we see that there are implicit dependencies between T_1 and T_4 as well as T_2 and T_5. But in fact, the detection of implicit dependencies is not as simple as this example shows.

Before starting to detect implicit dependencies, two auxiliary definitions should be given. They are used to represent the input and output task sets of one task set.

Definition 2 (Input and output task sets). *Let W be a one-length-loop-free workflow log over T. Let $T_s \subset T$:*

– *$\bullet T_s = \{t | \exists_{t' \in T_s} t \rightarrow_w t'\}$, and*
– *$T_s \bullet = \{t | \exists_{t' \in T_s} t' \rightarrow_w t\}$.*

Consider the workflow log shown in Table 1, the following conclusions can be drawn: $\bullet\{T_4\} = \{T_3\}$, $\bullet\{T_5\} = \{T_3\}$, and $\{T_1, T_2\}\bullet = \{T_3\}$.

Firstly, we try to detect implicit dependencies from a workflow log of a process model with a sub-WF-net similar to Fig.3 (b) and (g).

Theorem 1. *Let W be a one-length-loop-free workflow log over T. $T_w = \{t \in T | \exists_{\sigma \in W} t \in \sigma\}$. For any task t such that $t \in T_w$, if there are two tasks t_1 and t_2 such that $t_1 \in T_w$ and $t_2 \in T_w$ and $t_1 \rightarrow_w t$ and $t_2 \rightarrow_w t$ and $t_1 \#_w t_2$ and $\{t_1\}\bullet \neq \{t_2\}\bullet$, compute as follows:*

– *$X_w = \{(A, B) | A \subseteq T_w \wedge B \subseteq T_w \wedge t \in B \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_w b \wedge \forall_{a_1, a_2 \in A} a_1 \#_w a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_w b_2\}$, and*
– *$Y_w = \{(A, B) \in X_w | \forall_{(A', B') \in X_w} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B')\}$.*

Then for any $(A_1, B_1), (A_2, B_2)$ such that $(A_1, B_1) \in Y_w$ and $(A_2, B_2) \in Y_w$ and for any task a such that $a \in A_1$ and $a \notin A_2$, the following formula holds:

$$\nexists_{a' \in A_2} a' \parallel_w a \vee a' \succ_w a \Rightarrow \forall_{b' \in B_2 - \{a\}\bullet} a \mapsto_w b'.$$

Proof. See the sub-WF-net shown in Fig.4. Task t has two input places, i.e., p_1 and p_2. Because $a \rightarrow_w t$, there must be a workflow trace $\sigma = t_1 t_2 \ldots t_n$ and $i \in \{1, \ldots, n-1\}$ such that $t_i = a$ and $t_{i+1} = t$. After a is executed, there will be a token in p_1. In order to enable t, there must be a token in p_2 too. If $a' \succ a$, a' may have been executed before a is executed. There is a case that after a is executed, there is already a token in p_2. Hence after a is executed, t is enabled and can be executed. If $a' \parallel_w a$, the similar thing happens. Otherwise, t cannot be executed directly following a, thus here is a contradiction. So if $a' \not\succ_w a$ and $a' \nparallel_w a$, the relation $a \rightarrow_w t$ cannot be mined. In this case, there must be an arc connecting a and p_2 directly as the dotted arc shows. Thus, there must be an implicit dependency between a and b', i.e., $a \mapsto_w b'$. □

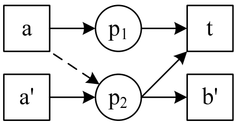

Fig. 4. Sub-WF-net for the proof of Theorem 1

Theorem 1 insures that once there is a place connecting two successive tasks in the mined model and the latter task has more than one input place, the latter task can always have chance to be executed directly following the former task.

Secondly, we try to detect implicit dependencies from a workflow log of a process model with a sub-WF-net similar to Fig.3 (c) to (f).

Theorem 2. *Let W be a one-length-loop-free workflow log over T.*
$T_w = \{t \in T | \exists_{\sigma \in W} t \in \sigma\}$.
(i) For any task t such that $t \in T_w$, if there are two tasks t_1 and t_2 such that $t_1 \in T_w$ and $t_2 \in T_w$ and $t \rightarrow_w t_1$ and $t \rightarrow_w t_2$ and $t_1 \parallel_w t_2$, compute as follows:

- $X_w = \{X \subseteq T_w | \forall_{x \in X} t \rightarrow_w x \wedge \forall_{x_1, x_2 \in X} x_1 \#_w x_2\}$, *and*
- $Y_w = \{X \in X_w | \forall_{X' \in X_w} X \subseteq X' \Rightarrow X = X'\}$.

Then for any task set Y such that $Y \in Y_w$, the following formula holds:

$$\forall_{a,b \in T_w} a \lhd_w b \wedge \exists_{y \in Y}(y \parallel_w b \vee y \succ_w b) \wedge \nexists_{y \in Y}(y \parallel_w a \vee y \succ_w a) \wedge t \gg_w a \Rightarrow t \mapsto_w a.$$

(ii) For any task t such that $t \in T_w$, if there are two tasks t_1 and t_2 such that $t_1 \in T_w$ and $t_2 \in T_w$ and $t_1 \rightarrow_w t$ and $t_2 \rightarrow_w t$ and $t_1 \parallel_w t_2$, compute as follows:

- $X_w = \{X \subseteq T_w | \forall_{x \in X} x \rightarrow_w t \wedge \forall_{x_1, x_2 \in X} x_1 \#_w x_2\}$, *and*
- $Y_w = \{X \in X_w | \forall_{X' \in X_w} X \subseteq X' \Rightarrow X = X'\}$.

Then for any task set Y such that $Y \in Y_w$, the following formula holds:

$$\forall_{a,b \in T_w} a \rhd_w b \wedge \exists_{y \in Y}(b \parallel_w y \vee b \succ_w y) \wedge \nexists_{y \in Y}(a \parallel_w y \vee a \succ_w y) \wedge a \gg_w t \Rightarrow a \mapsto_w t.$$

Proof. (i) See the sub-WF-net shown in Fig.5. After t is executed, there is one token in both p_1 and p_2. Because $t \gg_w a$, there must be a chance that one token appears in p_3 and a is enabled. At this time, if y_2 or y_2' is executed, a is disabled. In this case, b will be executed finally (see sub-condition 2 in the formula). Otherwise, if a is executed, y_2 and y_2' are both disabled (see sub-condition 3). The token in p_2 is left in the model, thus here is a contradiction. There must be an arc connecting p_2 and a directly as the dotted arc shows. Thus, there must be an implicit dependency between t and a, i.e., $t \mapsto_w a$.

(ii) The proof is similar to (i). □

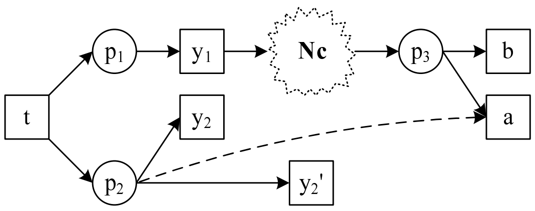

Fig. 5. Sub-WF-net for the proof of Theorem 2

Theorem 2 insures that once a task takes tokens from one of multiple parallel branches, it together with its parallel tasks must consume tokens from other branches too.

Finally, we try to detect implicit dependencies from a workflow log of a process model with a sub-WF-net similar to Fig.3 (a).

Theorem 3. *Let W be a one-length-loop-free workflow log over T. $T_w = \{t \in T | \exists_{\sigma \in W} t \in \sigma\}$. For any tasks a and b such that $a \in T_w$ and $b \in T_w$ and $a \rhd_w b$, compute as follows:*

- $X_w = \{(A,B) | A \subseteq T_w \wedge B \subseteq T_w \wedge (\forall_{a_i \in A} a \gg_w a_i \wedge b \not\gg_w a_i) \wedge (\forall_{b_j \in B} a \not\gg_w b_j \wedge b \gg_w b_j) \wedge \forall_{a_i \in A} \exists_{b_j \in B} b_j \lhd_w a_i \wedge \forall_{b_j \in B} \exists_{a_i \in A} a_i \lhd_w b_j \wedge \forall_{a_i, a_j \in A} a_i \parallel_w a_j \wedge \forall_{b_i, b_j \in B} b_i \parallel_w b_j\}$, *and*
- $Y_w = \{(A,B) \in X_w | \forall_{(A',B') \in X_w} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$.

For any (A,B) such that $(A,B) \in Y_w$, compute as follows:

- $A' = \{t \in T_w | t \notin A \wedge \exists_{b_j \in B} b_j \lhd_w t \wedge \exists_{a_i \in A} a_i \succ_w t\}$, *and*
- $B' = \{t \in T_w | t \notin B \wedge \exists_{a_i \in A} a_i \lhd_w t \wedge \exists_{b_j \in B} b_j \succ_w t\}$.

The following formulas holds:

- $\forall_{a_i \in A} \bullet \{a_i\} \subseteq \bullet B \cup \bullet B' \Rightarrow a \mapsto_w a_i$, *and*
- $\forall_{b_j \in B} \bullet \{b_j\} \subseteq \bullet A \cup \bullet A' \Rightarrow b \mapsto_w b_j$.

Proof. Here we only consider the simplest case. More complicated cases can be translated into this case by some way. See the sub-WF-net shown in Fig.6. If a is executed, there is a token in p_1. After a period of time, there is a token in p_3. Thus a_i is enabled ($a \gg_w a_i$) and b_j is still disabled. If b is executed, there will be one token in both p_1 and p_2. After a while, there will be one token in both p_3 and p_4. Thus a_i and b_j are both enabled. Because $b \not\gg_w a_i$, here is a contradiction. Task a_i can only be enabled after a is executed. Because $\bullet\{a_i\} \subseteq \bullet\{b_j\}$, there must be a place with its surrounding arcs directly connecting a and a_i as the dotted part of the figure shows. Thus, there must be an implicit dependency between a and a_i, i.e., $a \mapsto_w a_i$. □

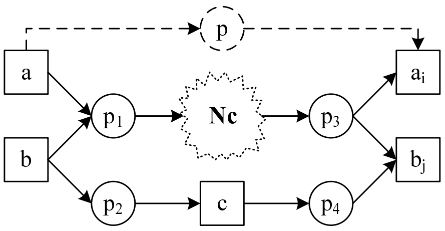

Fig. 6. Sub-WF-net for the proof of Theorem 3

Theorem 3 insures that if two exclusive tasks (i.e., involved in an OR-Join) lead to different sets of parallel branches and these two sets together with their tasks satisfy certain conditions (listed in Theorem 3), the mined WF-net is still sound.

5 Experimental Evaluation

A lot of experiments has been done to evaluate the proposed theorems in the previous section. Some of the experiments are listed below.

Fig.7 (a) shows an original WF-net. After applying α^+ algorithm on its log, the mined model is similar to Fig.7 (b) except for the two dotted arcs. Continue applying Theorem 1, $A \mapsto_w C$ is detected. Thus p_1 and p_2 should be merged together. At last, the fixed mined model will be the same as the original one.

Fig.8 shows the effect of applying Theorem 2. The WF-nets excluding the dotted arcs are mined by α^+ algorithm. The dotted arcs correspond to the

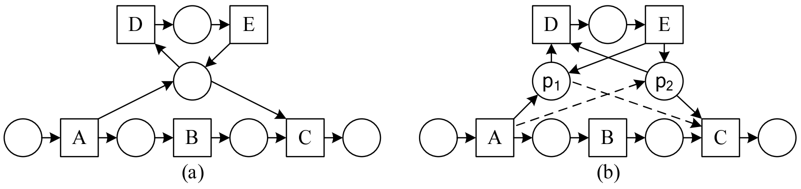

Fig. 7. Detect implicit dependencies by applying Theorem 1

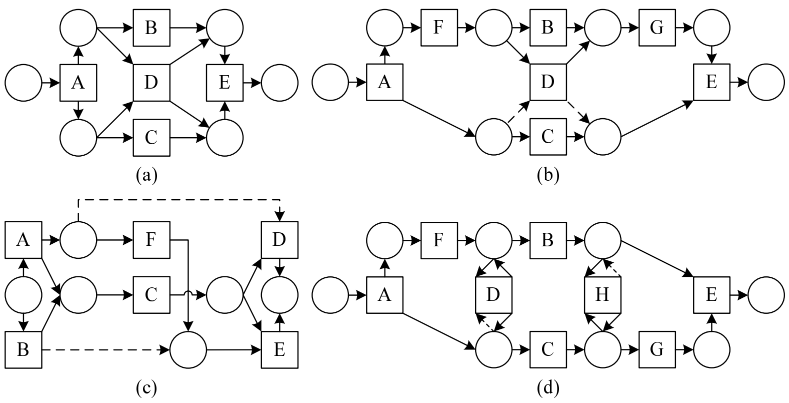

Fig. 8. Detect implicit dependencies by applying Theorem 2

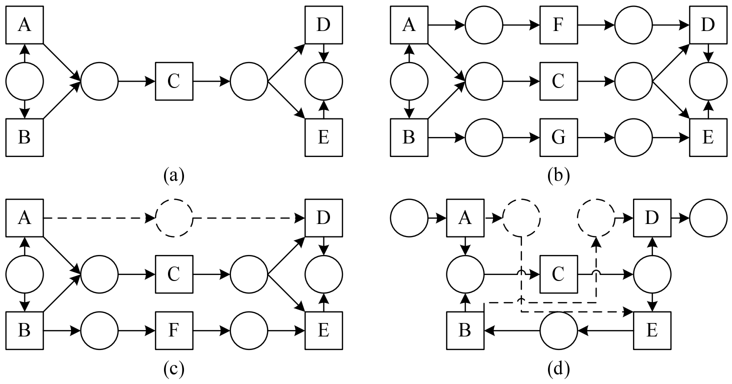

Fig. 9. Detect implicit dependencies by applying Theorem 3

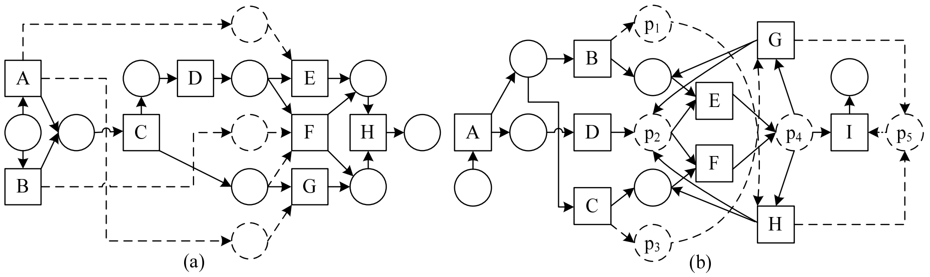

Fig. 10. Detect implicit dependencies by applying Theorem 2 and 3 as well as Theorem 1 and 3

detected implicit dependencies. Thus the WF-nets including the dotted arcs are the same as the original ones.

Fig.9 shows the effect of applying Theorem 3. All the implicit dependencies in the WF-nets are detected successfully from the logs.

Fig.10 (a) shows the effect of applying Theorem 2 and 3 successively. Fig.10 (b) shows the effect of applying Theorem 1 and 3 successively. The mined WF-nets with implicit dependencies are the same as the original ones.

These experimental results together with the formal proofs show that our theorems are powerful enough to detect implicit dependencies between tasks.

6 Related Work

Process mining is not a new topic, recently many researchers have done a lot of work on it [1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. For earlier related work, please refer to [13].

Schimm presents an approach on mining complete, most specific and minimal block-oriented workflow models from workflow logs [1], which is a special data mining. van der Aalst et al. present a common format for workflow logs, discuss the most challenging problems and review some of the workflow mining approaches available today [2]. In [3], the authors discuss the main issues around process mining, which include mining non-free-choice constructs. Medeiros et al point out that all the existing heuristic-based mining algorithms have their limitations [4]. After the presentation of α algorithm first proposed in [6], they classify the process constructs that are difficult for it to handle and discuss how to extend it. After that, they propose a new algorithm named α^+ to deal with so-called *short loops* [7] and implement it in a visual process mining tool named *EMit* [8]. In [9], Herbst et al. describe the main requirements for an interactive workflow mining system and how they derived them. Also, they implement the first prototype of an interactive workflow mining system called *ProTo*. In [10], they give an overview of the algorithms that were implemented within the *InWoLvE* workflow mining system. These algorithms can mine process models with multiple tasks having the same name. In [11, 12], Pinter et al. view the execution of an activity as a time interval, and present two new algorithms for synthesizing process models. Their approach can detect parallelism explicitly, which is similar to the approach proposed in [13]. Greco et al. investigate data mining techniques for process mining and provide an effective technique based on the rather unexplored concept of clustering workflow executions [14]. In [15], the authors propose an algorithm to discover workflow patterns and workflow termination states and then use a set of rules to mine the workflow transactional behavior.

However, none of those works discuss how to mine process models with non-free-choice constructs as well as detect implicit dependencies between tasks.

7 Conclusion and Future Work

In this paper, we try to detect implicit dependencies between tasks from event logs. Dependencies between tasks are classified into two classes, i.e., explicit and implicit ones. For mining process models with non-free-choice constructs, the detection of implicit dependencies is an important success factor. There are totally seven kinds of sound sub-WF-nets and they are grouped into three cases. Thus we propose three theorems for handling these cases and give their proofs. Experimental evaluations show that our theorems are suitable for detecting implicit dependencies between tasks.

Our future work will mainly focus on the following two aspects. Firstly, we will extend the α^+ algorithm to support our theorems, so that we will be able to mine WF-nets involving implicit dependencies directly by the extended algorithm. Secondly, we will further investigate other factors affecting the mining of process models with non-free-choice constructs.

Acknowledgement

This work is supported by the 973 Project of China (No. 2002CB312006) and the Project of National Natural Science Foundation of China (No. 60373011).

References

1. Schimm, G.: Mining exact models of concurrent workflows. Computers in Industry 53 (2004) 265–281
2. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering 47 (2003) 237–267
3. van der Aalst, W.M.P., Weijters, A.J.M.M.: Process mining: a research agenda. Computers in Industry 53 (2004) 231–244
4. de Medeiros, A.K.A., van der Aalst, W.M.P., Weijters, A.J.M.M.: Workflow Mining: Current Status and Future Directions. In: Meersman, R., et al. (Eds.): CoopIS/DOA/ODBASE 2003. LNCS, Vol. 2888. Springer-Verlag, Berlin Heidelberg (2003) 389–406
5. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems, and Computers 8(l) (1998) 21–66
6. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering 16(9) (2004) 1128–1142
7. de Medeiros, A.K.A., van Dongen, B.F., van der Aalst, W.M.P., Weijters, A.J.M.M.: Process Mining for Ubiquitous Mobile Systems: An Overview and a Concrete Algorithm. UMICS (2004) 151–165
8. van Dongen, B.F., van der Aalst, W.M.P.: EMiT: A Process Mining Tool. In: Cortadella, J. and Reisig, W. (Eds.): ICATPN 2004. LNCS, Vol. 3099. Springer-Verlag, Berlin Heidelberg (2004) 454–463
9. Hammori, M., Herbst, J., Kleiner, N.: Interactive Workflow Mining. In: Desel, J., Pernici, B., and Weske, M.(Eds.): BPM 2004. LNCS, Vol. 3080. Springer-Verlag, Berlin Heidelberg (2004) 211–226
10. Herbst, J., Karagiannis, D.: Workflow mining with InWoLvE. Computers in Industry 53 (2004) 245–264
11. Golani, M., Pinter, S.S.: Generating a Process Model from a Process Audit Log. In: van der Aalst, W.M.P., et al. (Eds.): BPM 2003. LNCS, Vol. 2678. Springer-Verlag, Berlin Heidelberg (2003) 136–151

12. Pinter, S.S., Golani, M.: Discovering workflow models from activities' lifespans. Computers in Industry 53 (2004) 283–296
13. Lijie, W., Jianmin, W., van der Aalst, W.M.P., Zhe, W., and Jiaguang, S.: A Novel Approach for Process Mining Based on Event Types. BETA Working Paper Series, WP 118, Eindhoven University of Technology, Eindhoven, 2004.
14. Greco, G., Guzzo, A., et al.: Mining Expressive Process Models by Clustering Workflow Traces. In: Dai, H., Srikant, R., and Zhang, C. (Eds.): PAKDD 2004. LNAI, Vol. 3056. Springer-Verlag, Berlin Heidelberg (2004) 52–62
15. Gaaloul, W., Bhiri, S., Godart, C.: Discovering Workflow Transactional Behavior from Event-Based Log. In: Meersman, R., Tari , Z. (Eds.): CoopIS/DOA/ODBASE 2004. LNCS, Vol. 3290. Springer-Verlag, Berlin Heidelberg (2004) 3–18