

Incremental Workflow Mining with Optional Patterns and Its Application to Production Printing Process

Weixiang Sun, Tao Li, Wei Peng, and Tong Sun

Abstract-- For today's business organizations, workflow models play important roles in analyzing the productivity, evaluating the performances and costs, optimizing the business operations, and supporting evolving services and products. Workflow mining, the process of empirically extracting structured process descriptions from a set of real executions, thus has attracted a lot of attention recently. However, there are several challenges that have not been fully addressed in the previous research: i) How can we mine process models with optional tasks? ii) How can we efficiently use new available workflow log data to incrementally update pre-existing workflow models or to complete previous partial process models? iii) How can we compare two different workflow models of similar organizations?

In this paper, we present our research efforts to address the above challenges. We present a workflow mining algorithm that is able to mine process models with optional tasks and propose an incremental workflow mining algorithm based on intermediate relationships such as ordering and independence. The intermediate relationships can also be used to facilitate the comparison of two process models. We successfully apply our algorithm to address the challenges in production printing workflow: how to update the process model according to the dynamic changes and how to compare the "designed" process and the "deployed and executed" one.

Index Terms—Process Model, Incremental, Optional tasks, Workflow mining

1. INTRODUCTION

Today's business organizations are characterized by global, dynamic, uncertain and error-prone environments. In order to compete in such contexts, business organizations are usually driven by explicit workflow process models [11]. Generally, a process is a set of tasks to be accomplished, where every task might have pre-requisites within the process that have to be fulfilled before execution [1]. Process models are typical representations of process instances and they specify the casual, sequential as well as dependency relationships among tasks.

In practice, process models¹ are crucial in determining the way in which the resources of an organization are used and they play an important role in analyzing the productivity, evaluating the performances and costs, optimizing the business operations, and supporting evolving services and products [2]. However, the non-transparency of cause-effect relationships that describe the

effects of process changes is the big barrier for maintaining and optimizing the process model [11]. In addition, there are discrepancies between the actual workflow processes and the processes perceived by the management [2].

For example, production printing is a typical application domain where the above problems exist. Production Printing Workflow aims to automatically minimize the deviations of processes, and intervention of operators, while making efficient use of resources. With advancement of printing technique, heterogeneous hardware devices, software, and even different media keep being added to production printing process. Moreover, since printing is a highly customized service, the manufacturing processes are modified along with various demands. Due to the ever-changing techniques, products and market environments, and the growing number of short-run jobs, it is time-consuming and complicated to construct the totally new printing workflow model from scratch. As a result, the pre-existing workflow model should be updated to reflect the new products added to the process model and the new requirements from the various clients, and be improved due to the poor resource utilization performance.

Although a lot of work has been reported on workflow mining [1, 2, 3, 4], there are several challenges that have not been fully addressed:

1. How can we mine process models with optional tasks?

For real applications, workflow models can contain optional tasks. Such process models cannot be extracted using existing algorithms as described in [1, 2]. For example, in a medical workflow model shown in Figure 1, the lab test is optional: the lab test is required only when a patient has diabetes and is 50 years old.

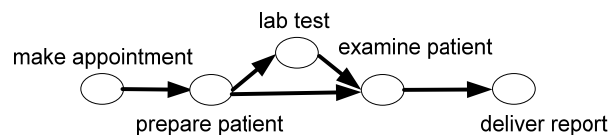


Figure 1: The medical workflow example with optional tasks.

2. How can we efficiently use new available workflow log data to incrementally update pre-existing workflow models or to complete previous partial process models? Incremental update and completion will allow us to take advantages of the legacy (pre-existing) models and human expertise while reflecting the current practice of the system. In addition, every time new type of service or product is provided or deployed, the process model needs to be updated.

This work is partially supported by a 2005 Xerox University Affairs Committee (UAC) Award and NSF CAREER grant IIS-0546280. Weixiang Sun, Tao Li, and Wei Peng are with School of Computer Science, Florida International University, Miami, FL 33199, USA (e-mail: wsun001, taoli, wpeng002@cs.fiu.edu). Tong Sun is with Adaptive & Smart Document System Lab, Xerox Innovation Group, Xerox Corporation, Webster, NY 14580, USA (e-mail: Tong.Sun@xeroxlabs.com).

¹ In this paper, we use workflow model and process model interchangeably.

3. How can we compare two different workflow models of similar organizations? In order to stay competitive, organizations want to learn from the successful processes of others to optimize the business operations.

In this paper, we present our research efforts to address the above challenges. Specifically, we first present a workflow mining algorithm that is able to mine process models with optional tasks; we then propose an incremental workflow mining algorithm based on intermediate relationships such as ordering and independence. The intermediate relationships can also be used to facilitate the comparison of two process models. We then apply our incremental workflow mining algorithm to address the challenges in production printing workflow: how to update the process model according to the dynamic changes and how to compare the “designed” process and the “deployed and executed” one.

The rest of the paper is organized as follows: Section 2 introduces the workflow mining algorithm that is able to mine process models with optional tasks, Section 3 proposes an incremental workflow mining algorithm based on intermediate relationships, Section 4 presents its application to production printing process, Section 5 surveys related work, and finally Section 6 presents our conclusions. A preliminary version of this paper has been appeared in [17].

2. WORKFLOW MINING ALGORITHM

In this section, we describe the workflow mining algorithm that is able to mine process models with optional tasks. Section 2.1 discusses the limitations of existing process mining algorithms, Section 2.2 introduces the basic concepts of workflow nets, Section 2.3 presents the details of the mining algorithm.

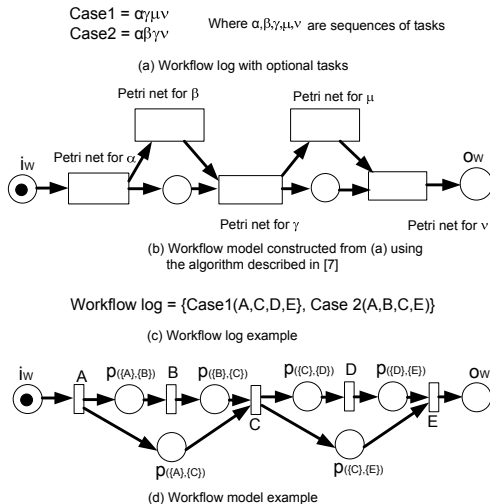


Figure 2: Problem with Workflow Mining Algorithm Presented in [7].

2.1 Limitations of Existing Algorithms

As we mentioned in Section 1, there are some limitations of existing algorithms as they cannot mine process models with optional tasks. Figure 2 illustrates some limitations of the algorithm proposed in [7]. Figure 2(a) describes a type of workflow logs with optional tasks. Figure 2(b) depicts the workflow model constructed from the workflow log in Figure 2(a) by the algorithm described in [7]. The constructed workflow model cannot produce the corresponding workflow log. Figure 2(c) and Figure 2(d) further illustrate this limitation by a concrete example. The workflow model shown in Figure 2(d) can not generate Case1(A,C,D,E) and Case2(A,B,C,E) in the workflow log shown in Figure 2(c). The workflow model shown in Figure 2(d) can only produce Case(A,B,C,D,E).

In other words, the workflow model with the optional pattern² cannot be mined using the algorithm described in [7]. However these patterns are very important in practice. For example, in a travel agency, when a client find that he/she is not interested in the information provided by an agent, the system should allow him/her to leave, then the remaining tasks will become optional. The places where the client decides to leave are important for improving the business process because those places might indicate where the service should be improved.

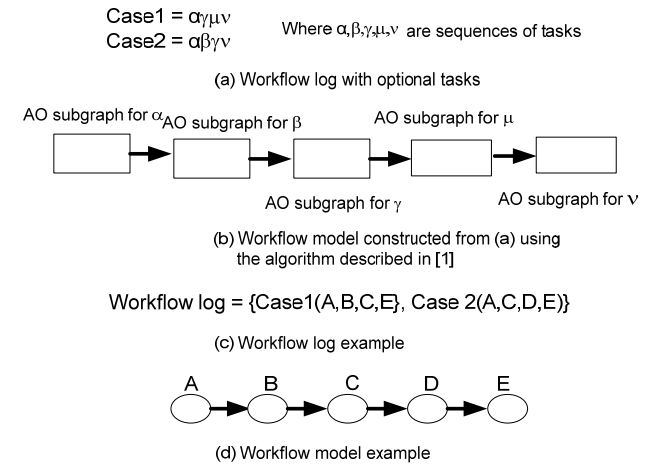


Figure 3: Problem with Workflow Mining Algorithm Presented in [1].

For the probabilistic workflow mining algorithm presented in [1], there are similar problems as shown in Figure 3: the workflow models with optional tasks can not be mined.

In the rest of the section, we will propose a workflow mining algorithm to address the above problems.

2.2 Workflow Nets

In workflow models, there exist different relationships between the tasks: *sequential*, *parallel*, and *mutually exclusive*. Some tasks can be *cyclic* or *optional*. Workflow

² In this paper, the optional pattern means some tasks are optional.

models should capture these relationships between the tasks. In this paper, we choose Petri nets [8, 9, 10] as the representation of workflow models. The reasons for specifying the workflow models by using Petri nets are stated in [12]: Petri nets are formal, have associated analysis techniques, and are state-based rather than event-based. There are many classes of Petri nets [8, 9, 10]. In the following, we will give a brief introduction of Petri Net used in our paper.

DEFINITION 2.1. (Petri Nets) A Petri net is a tuple (P, T, F) where:

- P is a finite set of places,
- T is a finite set of transitions such that $P \cap T = \emptyset$, and
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation.

DEFINITION 2.2. (Marking of a Petri net) Places in a Petri net may contain so-called tokens. The function from the place set P of a Petri net N to the natural number is called a Marking (M) of N . For any place p in P , $M(p)$ denotes the number of tokens in this place.

DEFINITION 2.3. (Preset, Postset) Let $N=(P, T, F)$ be a Petri net. For any $x \in P \cup T$:

- $\cdot x = \{y \in P \cup T \mid (y, x) \in F\}$ (preset of x) and
- $x \cdot = \{y \in P \cup T \mid (x, y) \in F\}$ (postset of x)

DEFINITION 2.4. (Firing rule) Let $N=(P, T, F)$ be a Petri net, M be a marking of N , and $t \in T$:

- M enables t iff $\cdot t \leq M$
- M_1 is reachable from M by firing t (denoted by “ Mt ”) iff M_1 enables t and $M_1 = M - \cdot t + t \cdot$

DEFINITION 2.5. (Firing sequence) Let $N=(P, T, F)$ be a Petri net and M_0 be an initial marking of N . A sequence $\sigma \in T^*$ (T^* is the universal set of transition sequences) is called a firing sequence of (N, M_0) iff there exist markings $M_1, \dots, M_n$ and transitions $t_1, \dots, t_n \in T$ such that $\sigma = t_1 \dots t_n$ and for all i with $0 \leq i < n$: t_{i+1} is enabled by M_i and $M_{i+1} = M_i - \cdot t_{i+1} + t_{i+1} \cdot$

Let the transition set T in Petri nets represent a set of tasks. The firing sequence of Petri nets [8, 9, 10] is a workflow execution trace. Since there is no corresponding construct in Petri Nets for each kind of hidden states (and-split, and-join, or-split and or-join [11]), we model them as a piece of Petri Nets shown in Figure 4. The added transitions do not represent real tasks. They are just used to represent the hidden states (split and join tasks). So the transitions in the Petri net used in our algorithm include not only tasks appearing in workflow log, but also some hidden states.

Workflow nets (modeling the control flow of a workflow) are then defined as follows:

DEFINITION 2.6. (Workflow nets [7]) Let $N=(P, T, F)$ be a Petri net and $\bar{t}$ a fresh identifier not in $P \cup T$. N is a workflow net (WF-net) iff:

- *object creation*: P contains an input place i such that $\cdot i = \emptyset$
- *object completion*: P contains an output place o such that $o \cdot = \emptyset$
- *connectedness*: $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ is strongly connected.

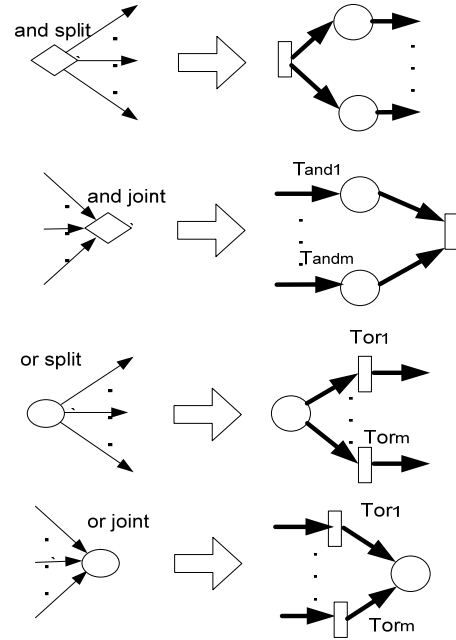


Figure 4: Modelling and-split, and-join, or-split and or-join with Petri Nets

2.3 Workflow Mining Algorithm

The workflow model in our algorithm will be represented by workflow nets. There are some assumptions about our workflow nets $N=(P, T, F)$:

- For all $x \in P \cup T$, there is no sequence $x_0, \dots, x_n$ so that $x_i \in P \cup T$ and $(x_i, x_{i+1}) \in F$ where $0 \leq i < n$ and $x = x_0 = x_n$.
- For all $x \in P \cup T$, if $|\cdot x| > 1$, there exists $y \in P \cup T$ so that there is a path from any element in $\cdot x$ to y .
- If $x\sigma y$ and $x\alpha y$ are two paths in N , z is any node of workflow net and there exists a path from any node in a sequence $\sigma\alpha y$ to z , then there is a path from y to z or a path from z to y .

The first condition makes sure that the workflow net has no cycle. The last two conditions essentially enforce nesting of threads. These constraints on workflow graph are often adopted in workflow mining literatures [1, 14, 15, 16].

The input to our workflow mining algorithm includes an ordering oracle O and an independence oracle I . $O(T_1, T_2)$ describes the ordering relationship between T_1 and T_2 . For example, if T_1 always happens before T_2 , $O(T_1, T_2)$ will be true. If T_1 and T_2 can not happen in one workflow trace, $O(T_1, T_2)$ should be exclusive. $I(T_1, T_2, T)$ describes if T_1 and T_2 are independent conditioned on task T . Details about ordering oracle and independence oracle are discussed in [1].

Our workflow mining algorithm is built upon [1]. The main difference is that: in our algorithm, optional patterns can be mined and the workflow model is represented by the workflow net. When we add a hidden state, we will add a piece of Petri net representing this hidden state. The challenge is how to mine the optional pattern. Details of workflow mining algorithm will be discussed in Appendix A.

2.3.1. Optional Pattern Discovery

In this section, we will describe how the optional pattern is discovered by our algorithm. With only ordering and independence oracles, we cannot find optional patterns. We need more information from the workflow log. When the tasks in *CurrentBlanket* are added to the workflow net, these tasks will be removed. When we get *CurrentBlanket*, the workflow traces in current workflow log should start with *CurrentBlanket* if there is no optional pattern. If there is workflow traces starting with the tasks not included in *CurrentBlanket*, it is possible that such traces contain noise.

We assume a threshold for the frequency of these workflow traces. If the frequency of these workflow traces is higher than the threshold, there exists an optional pattern. We record them in *OptionalSet* when the algorithm **HiddenSplits** is executed. Because we assume that the workflow structure is nested, it cannot reach internal points of the nested structure from outside. Every time when we split the *CurrentBlanket*, we will check if *CurrentBlanket* is contained in *OptionalSet*. If it is, we will obtain an optional pattern and register such information in *OptionalSet*. At the end of the algorithm **LearnWorkflowNet**, we will execute the algorithm **AddOption** shown in Figure 17 to add optional patterns to the constructed workflow net.

3. INCREMENTAL WORKFLOW MINING ALGORITHM

3.1. The High-Level Process of Incremental Workflow Mining

Generally, the workflow mining is divided into two phases (shown in Figure 5) [2]:

1. Workflow log is input to the first phase, and data mining techniques are applied on them to get the relationships between activities, such as: ordering and independence [1], dependency graph [6], Log-based ordering relations [7] and so on.

2. Workflow model is built based on the relationships between activities.

In practice, there are two scenarios for incremental workflow mining:

1. If there is a pre-existing workflow model and we have new workflow log data, how can we update workflow model?
2. Domain experts usually are able to construct a partial process model based on their domain knowledge. How can we complete such partial workflow models with the workflow log data?

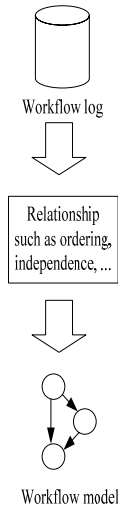


Figure 5: Workflow mining process

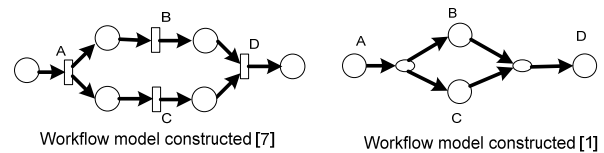


Figure 6: How to compare different representations of the same workflow log data

The natural way to handle the incremental workflow mining, when new workflow log data are available, is: first construct a new workflow model from new workflow log data, and then compare it with the pre-existing workflow model to get the difference, and finally update/complete the pre-existing workflow model using the difference.

The above solution is done at the workflow model level. However, there are some problems for this solution:

- For the same workflow log data, we can build different workflow models based on different workflow mining methods. It is hard to automatically compare these two different process models. An Example is shown in Figure 6.
- The hidden states (such as split state and join state) depend on the relationship extracted during the first phase. It is difficult to find new hidden states when comparing two different workflow models. In

addition, the relationships extracted are different and mined workflow models will have different hidden states. It is hard to figure out the difference between these hidden states and thus difficult to merge them.

To address the difficulties mentioned above, we need to perform the incremental workflow mining at the intermediate relationship level.

3.2. Incremental Workflow Mining Algorithm Description

In this section, we will give a detailed description of the incremental workflow mining algorithm shown in Figure 7.

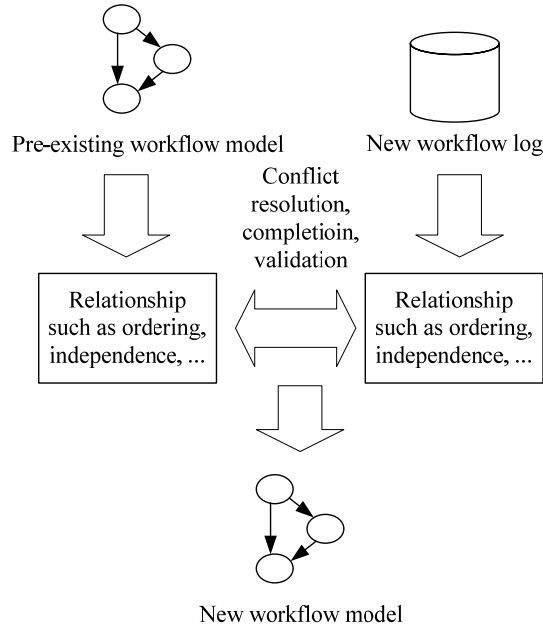


Figure 7: Incremental workflow mining process

The algorithm is done based on the intermediate relationship. The intermediate relationship will be extracted from both the pre-existing workflow model and new workflow log data. In this paper, we use our workflow mining algorithm to mine the intermediate relationship from new workflow log data³.

Generally workflow mining algorithm does not provide procedures to extract the intermediate relationships from the workflow models. We provide algorithms as in Figure 8 and Figure 9 to mine the ordering relationship and independence relationship respectively from pre-existing workflow nets. When we obtain the intermediate relationships from the pre-existing workflow model and new workflow log, they maybe inconsistent and we need to remove these inconsistencies. The algorithm **OrderingForIncremental** shown in Figure 10 illustrates

how to combine the different ordering relationships. The algorithm **IndependenceForIncremental** shown in Figure 11 describes how to combine the different independence relationships. We define different conflict resolutions for different purposes: completion, update and comparison in the above two algorithms. We reuse our workflow mining algorithm by replacing Ordering and Independence oracles with **OrderingForIncremental** and **IndependenceForIncremental** and will produce completed workflow models or updated workflow models.

Algorithm LearnOrdering

Input T_1, T_2 , tasks;
 N , Workflow Net;
Output result, {true, exclusive, undefined}
1. if there are two sequences: $\{x_1, \dots, x_m\}, \{y_1, \dots, y_n\}$ where $\forall 1 \leq i \leq m: (x_i, x_{i+1}) \in F(N)$ and $\forall 1 \leq j \leq n: (y_j, y_{j+1}) \in F(N)$ and $\{x_2, \dots, x_m\} \cap \{y_2, \dots, y_n\} = \emptyset, x_1 = y_1 \in T(N)$ and $x_m = T_1, y_n = T_2$
2. return true
3. if there is one sequence: $\{x_1, \dots, x_m\}$, where $\forall 1 \leq i \leq m: (x_i, x_{i+1}) \in F(N)$ and $x_1 = T_1, x_m = T_2$
4. return true
5. if there are two sequences: $\{x_1, \dots, x_m\}, \{y_1, \dots, y_n\}$ where $\forall 1 \leq i \leq m: (x_i, x_{i+1}) \in F(N)$ and $\forall 1 \leq j \leq n: (y_j, y_{j+1}) \in F(N)$ and $\{x_2, \dots, x_m\} \cap \{y_2, \dots, y_n\} = \emptyset, x_1 = y_1 \in P(N)$ and $x_m = T_1, y_n = T_2$
6. return exclusive
7. return undefined

Figure 8: **LearnOrdering**: an algorithm to learn the ordering relationship from the workflow net

Algorithm LearnIndependence

Input T_1, T_2, T_3 : a task;
 N , Workflow net
Output Independence, boolean
1. If Dseparation(T_1, T_2, T_3, N)
2. Return true;
3. If $(\exists x: (\text{SureAncestor}(x, T_3, N) \text{ and } \text{Dseparation}(T_1, T_2, T_3, N)))$
4. Return true;
5. If $\text{SuerAncestor}(T_1, T_3, N) \text{ or } \text{SureAncestor}(T_2, T_3, N)$
6. Return true;

Algorithm SureAncestor

Input T_1, T_2 : tasks
 N , workflow net
Output result, boolean
1. Ancestor = { $t \mid \exists x_1, \dots, x_m$, where $\forall 1 \leq i \leq m: (x_i, x_{i+1}) \in F(N)$ and $x_1 = t, x_m = T_2$ }
2. If $(\forall x \in \text{Ancestor}: (\exists x_1, \dots, x_m$, where $\forall 1 \leq i \leq m: (x_i, x_{i+1}) \in F(N)$ and $x_1 = x, x_m = T_1) \wedge (\text{Dseparation}(x, T_2, T_1) \text{ or } (\exists y_1, \dots, y_n$, where $\forall 1 \leq i \leq m: (y_i, y_{i+1}) \in F(N)$ and $y_1 = T_2, y_m = x)))$
3. Return true

Algorithm DSeparation

Input T_1, T_2, T_3 : tasks
 N , workflow net
Output separation, boolean
1. if $\exists x_1, \dots, x_m: (\forall 1 \leq i \leq m: (x_i, x_{i+1}) \in F(N) \wedge (x_i \neq T_3)) \wedge (x_1 = T_1) \wedge (x_m = T_2) \wedge (x_m \neq T_3)$
2. Return false
3. if $\exists x_1, \dots, x_m, y_1, \dots, y_n, z_1, \dots, z_k: (\forall 1 \leq i \leq m: (x_i, x_{i+1}) \in F(N)) \wedge (\forall 1 \leq j \leq n: (y_j, y_{j+1}) \in F(N)) \wedge (\forall 1 \leq k \leq k: (z_i, z_{i+1}) \in F(N)) \wedge (x_1 \neq T_1) \wedge (y_1 = T_2) \wedge (z_k = T_3)$
4. Return false
5. Return true

Figure 9: **LearnIndependence**: an algorithm to learn the independence relationship from the workflow net

³ It should be pointed that: existing workflow mining algorithms in [1, 7] can also be used in our incremental mining framework.

Algorithm OrderingForIncremental
 Input T_1, T_2 : tasks
 O , ordering oracle for new workflow log;
 N , a pre-existing workflow net;
 OP , operation
 Output R , ordering relationship between T_1, T_2
 1. $result1 \leftarrow \text{LearnOrdering}(T_1, T_2, N)$
 2. $result2 \leftarrow O(T_1, T_2)$
 3. If OP is Completion
 4. If $result1$ is undefined
 5. return $result2$
 6. else
 7. return $result1$
 8. If OP is Update
 9. If $result2$ is undefined
 10. return $result1$
 11. else
 12. return $result2$
 13. if OP is Comparison
 12. if $result1 \neq result2$
 13. print $T_1, T_2, result1, result2$
 14. return $result1$

Figure 10: **OrderingForIncremental**: an algorithm to combine the ordering relationship gained from the workflow net and the workflow log data

Algorithm IndependenceForIncremental
 Input T_1, T_2, T_3 : tasks
 I , ordering oracle for new workflow log;
 N , a pre-existing workflow net;
 OP , operation
 Output R , independence relationship between T_1, T_2, T_3
 1. $result1 \leftarrow \text{LearnIndependence}(T_1, T_2, T_3, N)$
 2. $result2 \leftarrow I(T_1, T_2, T_3)$
 3. if T_1, T_2 or $T_3 \notin T(N)$
 4. return $result2$
 5. if T_1, T_2 or T_3 doesnot appear in new workflow log
 6. return $result1$
 7. If OP is Completion
 8. return $result1$
 9. If OP is Update
 10. return $result2$
 11. if OP is Comparison
 12. if $result1 \neq result2$
 13. print $T_1, T_2, T_3, result1, result2$
 14. return $result1$

Figure 11: **IndependenceForIncremental**: an algorithm to combine the independence relationship gained from the workflow net and the workflow log data.

From the above discussions, we observe that the comparison of intermediate relationships is an important component for our incremental workflow algorithm. This is helpful for us to solve the following scenario: when we have different workflow models for similar organizations, how to compare them to get the difference for improving the properties of one workflow model. We can log the difference when we build the new workflow model in our incremental workflow mining algorithms as shown in Figure 10 and Figure 11. The difference can then be used for improving the workflow models.

3.2. Incremental Workflow Mining Tool

We build a software tool in Java using JCreator 2.5 IDE under the Windows XP operating system. The functionalities of the tool are described as follows:

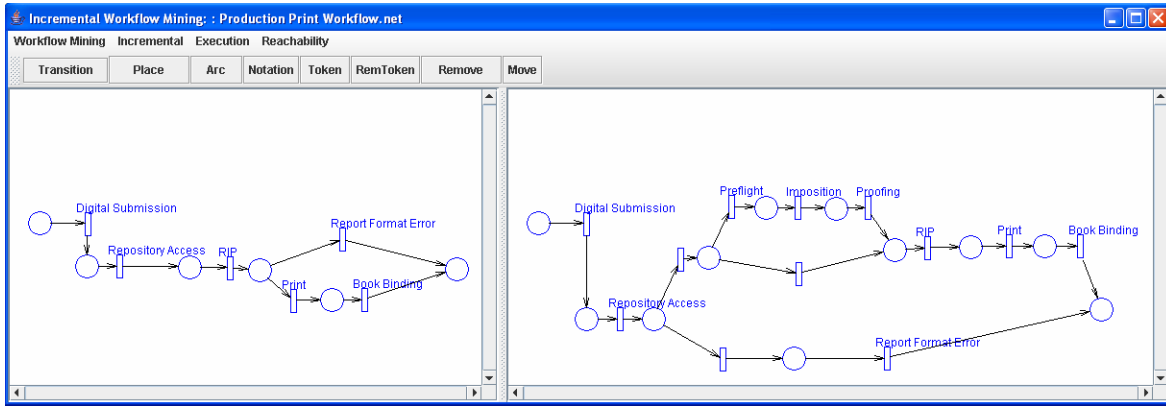
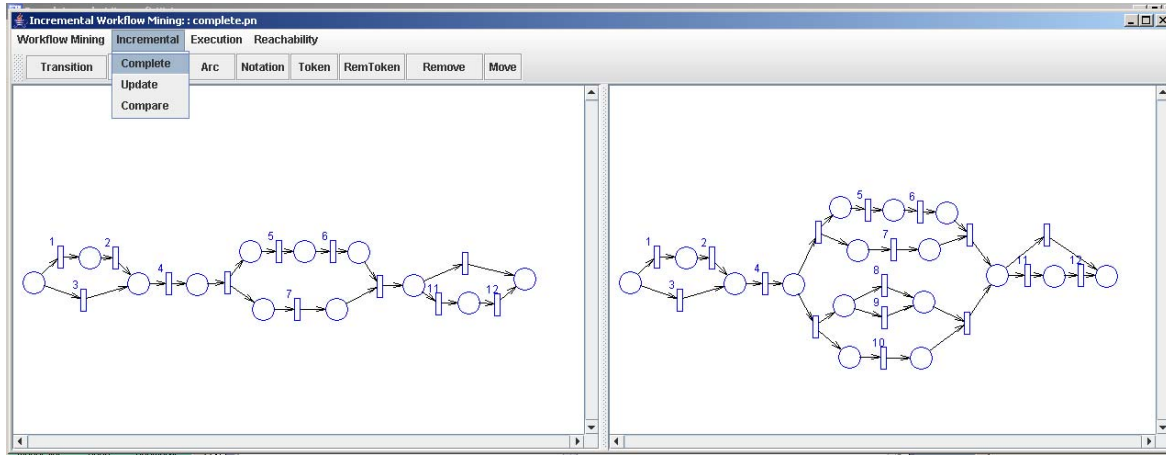
- The tool allows the construction of the workflow nets and the generation of the workflow log when executing the workflow net.
- The tool is able to complete the workflow net, i.e. it loads a pre-existing workflow net and the new workflow log, and then completes the pre-existing workflow net based on the new workflow log data.
- The tool can update the workflow net based on the new workflow log.
- The tool can compare two workflow nets which describe similar workflow processes and produce the difference as the output.
- The tool can construct the reach-ability tree that can be used to formally analyze the workflow net.

Our incremental workflow mining tool only logs the difference when building the new workflow model. The difference can then be presented to domain experts to help them make decisions for improving the business process. In practice, domain knowledge is important for comparing two workflow models.

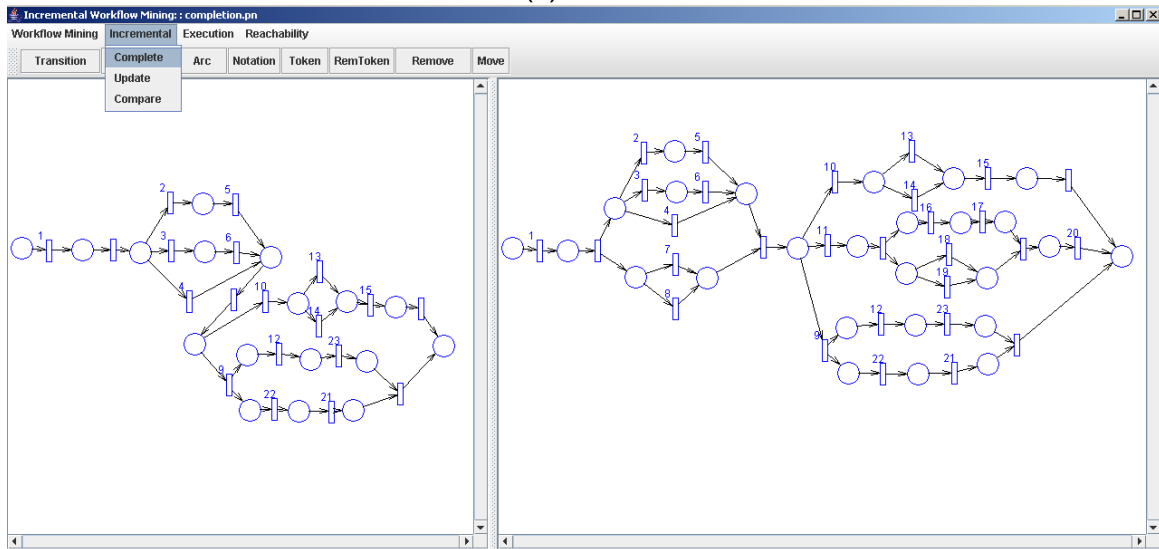
3. APPLICATION TO PRODUCTION PRINTING PROCESS

Production Printing Workflow (PPW) is a high-volume and high-speed printing process where various printing related tasks namely pre-press, press and post-press cooperate with each other and perform collectively for producing printed materials like books, catalogs, manuals, financial statements, collaterals, etc. Pre-press refers to a set of procedures that process documents before printing, such as composition, preflight, imposition etc. Press is the actual procedure of printing documents on a specific media. And post-press is the procedure involves finishing the printed materials, such as cutting, folding, and binding, etc. Traditional production printing workflows are more of human-controlled manufacturing processes that are encompassed by a complex set of hardware machineries and manual interventions across all tasks. Nowadays, the entire printing business is undergoing a major paradigm shift into a marketplace centered with more automated digital workflows, in which heterogeneous devices and increasing number of software applications seamlessly interoperate with each other. The ever-increasing challenges for today's production printing business include:

- The hybrid workflows which include both traditional press and digital press.
- The dynamic environments resulting from frequently product upgrades, emerging new technologies and capabilities.
- The heterogeneous offerings from various vendors.
- A growing number of short-run jobs require more emphasis on workflow efficiency.

Figure 12: **Practical Example:** Update of Production Printing Workflow Net for digital book printing.

(a)



(b)

Figure 13: **Production Printing Workflow Experiments:** complete the pre-existing workflow net in the left side with the workflow log data and the new workflow net is in the right side. The transitions with the number represent the real tasks in production printing workflow.

Process model plays an important role in representing the end-to-end production printing workflow for workflow analysis, integration, automation, and transformation purposes. However, the challenges remain as:

- When product upgrades are performed and/or new products are introduced, what the impacts are on the existing process model, how to keep the process model seamlessly updated according to these dynamic changes
- What are the gaps between the "designed" process model and the "deployed and executed" process model? The answers to this question may provide insights regards run-time workflow efficiency and help to identify the potential areas for process improvement

In this section, we will try to show how incremental workflow mining algorithm will help us address these issues. Usually, product upgrades are performed before changing the process model.

After the product upgrade, the workflow is changed in order to take advantage of the new product. The workflow log will keep track of the changes. We can update the workflow model based on new workflow log by using the incremental workflow mining algorithm. The "deployed and executed" process model will be recorded in the workflow log. Our incremental workflow mining algorithm can find out the difference between the "designed" process model and the existing workflow log produced by the "deployed and executed" process model.

In the following, we describe how we update the workflow model in digital book printing using our algorithm. We do the other experiments in production printing workflow.

Digital book printing is one of growing areas that digital technologies penetrate traditional press with promising on-demand capabilities and short-run efficiency. In this section, we will show how to update the existing digital book printing workflow model based on the new workflow log. The existing workflow model is shown in the left side of Figure 12 where *Digital Submission*, *Repository Access* and *RIP*⁴ are the pre-press tasks, *Printing* is the press task and *Booking Binding* is the post-press task.

The existing printing workflow model only reproduced the books from pre-mastered digital file. Now in order to reduce human-controlled manufacturing processing, more pre-press tasks including *Preflight*, *Imposition*, and *Proofing* are introduced are required to handle more untreated digital files. The errors should be reported immediately after *Repository Access*, the performance would be increased. The workflow model should be updated to reflect the changes. We got the workflow log from the Xerox Company and update the workflow model using the new workflow log. The new workflow net is

shown in the right side of Figure 12. The new model can be analyzed to see what kinds of changes are made to the workflow model in practice and whether these changes are reasonable. From this example, we can see the incremental workflow mining can be used to reflect the practical changes which are not reflected in existing workflow model.

We perform two other experiments about the production printing workflow as shown in Figure 13(a) and Figure 13(b). For saving the place, we do not mention the specific names of the tasks. For each experiment, the workflow net as shown in the left side of Figure 13 represents the pre-existing workflow model. We got the real production printing workflow log from the Xerox Company. The goal of the experiments is to complete the pre-existing workflow nets with the new workflow log data. The workflow nets shown in the right side of Figure 13 are generated as the results of the workflow model completion.

4. RELATED WORK

The first algorithm for mining workflow logs is introduced in [6]. Its main task is to find a workflow graph generating events appearing in a given workflow log. [7] It discovers from event logs the process model represented by Petri Nets. And [3] proposes process mining algorithms by clustering workflow traces. Recently, [1] introduces a workflow mining algorithm based on a coherent probabilistic model. A detailed survey on the current work in workflow mining, or process mining is given in [2]. The development of Woflan [13] demonstrates that workflow model specified as Petri nets can be analyzed.

Our workflow mining algorithm is able to produce the workflow model specified by Petri nets. As we discussed in Section 1, although a lot of work has been reported on workflow mining, there are several challenges, such as mining process models with optional tasks, incremental workflow mining, comparisons of workflow models, have not been fully addressed. To the best of our knowledge, our research effort made the very first step to address those challenges.

5. CONCLUSION

Today's business organizations are characterized by global, dynamic, uncertain and error-prone environments. In order to compete in such contexts, workflow models play important roles in analyzing the productivity, evaluating the performances and costs, optimizing the business operations, and supporting evolving services and products. Although a lot of research work has been reported in workflow mining recently, there are several challenges, such as mining process models with optional tasks, incremental workflow mining and comparisons of workflow models, have not been fully addressed in previous research. In this paper, we present our research efforts that attempt to address the above challenges. We present a workflow mining algorithm that is able to mine

⁴ RIP is the process of translating the digital information about fonts and graphics that describes the appearance of the file into the image that the printer can output.

process models with optional tasks and propose an incremental workflow mining algorithm based on intermediate relationships such as ordering and independence. The intermediate relationships can also be used to facilitate the comparison of two process models. Experiments are conducted to illustrate the effectiveness of our algorithms on example data derived from real world applications.

REFERENCES

- [1] R. Silva, J. Zhang and J. G. Shanahan, "Probabilistic workflow mining", *Proc. of the 11th ACM SIGKDD international conference on Knowledge discovery in data mining*, pages:275-284, 2005.
- [2] W. V. D. Aalst and A. Weijters, "Process mining: a research agenda", *Computers and Industry*, 53:231-244, 2004.
- [3] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, "Mining expressive process models by clustering workflow traces", *Proc. Of the 8th PAKDD*, 2004.
- [4] J. Herbst and K. Karagiannis, "Workflow mining with InWoLvE", *Computers and Industry*, 53:245-264, 2004.
- [5] J. E. Cook and A. L. Wolf, "Software process validation: quantitatively measuring the correspondence of a process to a model", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8(2):147-176, 1999.
- [6] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs", *Proc. of 6th international Conference on Extending Database Technology*, pages 469-483, 1998.
- [7] W. V. D. Aalst, T. Weijters and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs", *IEEE Transactions on Knowledge and Data Engineering*, 16(9): 1128 - 1142, 2004.
- [8] J. Desel and J. Esparza, "Free Choice Petri Nets", *volume 40 of Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press, Cambridge, UK, 1995.
- [9] W. Reisig and G. Rozenberg, editors, "Lectures on Petri Nets I: Basic Models", *volume 1491 of Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1998.
- [10] T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, 77(4):541-580, April 1989.
- [11] W. V. D. Aalst and K. van Hee, "Workflow Management: Models, Methods and Systems", *MIT Press*, 2002.
- [12] W. V. D. Aalst, "Three Good Reasons for Using a Petri-net-based Workflow Management System", *In S. Navathe and T. Wakayama, editors, Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, Cambridge, Massachusetts, Nov. 14-15, 1996, pages 179-201. 1996.
- [13] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst, "Diagnosing workflow Processes using Woflan", *The Computer Journal*, 44(4):246-279, 2001.
- [14] G. Schimm, "Generic Linear Business Process Modeling", *In S.W. Liddle, H.C. Mayr, and B. Thalheim, editors, Proceedings of the ER 2000 Workshop on Conceptual Approaches for E-Business and The World Wide Web and Conceptual Modeling*, volume 1921 of LNCS, pages 31-39. Springer-Verlag, Berlin, 2000.
- [15] J. Herbst and D. Karagiannis, "Workflow mining with InWoLvE", *Computers and Industry*, 53:245-264, 2004.
- [16] G. Schimm, "Mining Exact Models of Concurrent Workflows", *Computers and Industry*, 53:265-281, 2004.
- [17] Weixiang Sun, Tao Li, Wei Peng, and Tong Sun, "Incremental Workflow Mining with Optional Patterns", *IEEE International Conference on Systems, Man, and Cybernetics*, 2006.

Algorithm LearnWorkflowNet

Input O , an ordering oracle for a set T of tasks;
 I , an independence oracle for T ;
Output N , workflow net

1. $N=(O,O,O)$ and G is a graph (V,E) where $V=T$ and $E=O$
2. $E(G) \leftarrow E(G) \cup \{(t_1,t_2) | O(t_1,t_2)=\text{true and } O(t_2,t_1)=\text{false}\}$
3. $\text{CurrentBlanket} = \{t | \forall t_1 \in T: (t_1,t) \notin E(G)\}$
4. $T(N) \leftarrow T(N) \cup \text{CurrentBlanket}$
5. $\text{OptionalSet} \leftarrow \emptyset$
6. $(N, \text{SplitNode}, \text{OptionalSet}) \leftarrow \text{HiddenSplits}(N, \text{CurrentBlanket}, O, \text{OptionalSet})$
7. if SplitNode is transition
8. Create a new place p
9. add arc $(p, \text{SplitNode})$ to N
10. $V(G) \leftarrow V(G) - \text{CurrentBlanket}$
11. While $V(G) \neq \emptyset$
12. $\text{NextBlanket} \leftarrow \text{GetNextBlanket}(\text{CurrentBlanket}, G, O, I)$
13. $T(N) \leftarrow T(N) \cup \text{NextBlanket}$
14. $\text{Ancestors} \leftarrow \text{Dependencies}(\text{CurrentBlanket}, \text{NextBlanket}, O, I)$
15. $(N, \text{OptionSet}) \leftarrow \text{InsertLatents}(N, \text{CurrentBlanket}, \text{NextBlanket}, \text{Ancestors}, O, \text{OptionalSet})$
16. $G \leftarrow G - \text{NextBlanket}$
17. Let CurrentBlanket be the subset of T whose elements donot have a child in N
18. $(N, \text{JoinNode}) \leftarrow \text{HiddenJoins}(N, \text{CurrentBlanket}, O)$
19. if JoinNode is transition
20. Create a new place p
21. add arc $(\text{JoinNode}, p)$ to N
22. $N \leftarrow \text{addOption}(N, \text{OptionalSet})$
23. Return N

Figure 14: **LearnWorkflowNet**: an algorithm for Learning Workflow Net.

Appendix A: Brief Description of Workflow Mining Algorithm

The main algorithm **LearnWorkflowNet** is described in Figure 14. The algorithm makes reference to the algorithms given in Figures [15-18]. The algorithm iteratively adds nodes (places and transitions) to a partially-built workflow net in a specific order. First, we build the ordering relationship graph G based on the

Algorithm GetNextBlanket

Input CurrentBlanket , a set of tasks;
 G , a DAG encoding ancestral relationships;
 O , an ordering oracle;
 I , an independence oracle;
Output NextBlanket , a subset of the tasks in G

1. $E(G) \leftarrow E(G) - \{(t_1,t_2) \in E(G) | \forall t \in \text{CurrentBlanket}: O(t,t_1) \neq \text{exclusive and } O(t,t_2) \neq \text{exclusive and } I(t_1,t_2)\}$
2. $\text{NextBlanket} = \{t | \forall t_1 \in T: (t_1,t) \notin E(G)\}$
3. Return NextBlanket

Figure 15: **GetNextBlanket**: an algorithm to identify the next set of tasks to be added.

Algorithm **InsertLatents**
 Input N, a Petri Net;
 CurrentBlanket, NextBlanket, two sets;
 AG, an ancestral DAG;
 O, an ordering oracle;
 OS, optional set
 Output N, a Petri Net
 OS, optional set

1. For every task $t \in \text{NextBlanket}$
2. Siblings $\leftarrow \{t_1 \in \text{NextBlanket} \mid \exists t_2 \in V(\text{AG}); (t_2, t) \in E(\text{AG}), (t, t_1) \in E(\text{AG})\}$
3. AncestralSet $\leftarrow \{t_1 \in V(\text{AG}) \mid \forall t_2 \in V(\text{Siblings}): (t_1, t_2) \in E(\text{AG})\}$
4. (N, JoinNode) $\leftarrow \text{HiddenJoins}(N, \text{AncestralSet}, O)$;
5. (N, SplitNode, OS) $\leftarrow \text{HiddenSplits}(N, \text{Siblings}, O, OS)$;
6. If JoinNode is place and SplitNode is transition or reversely
7. add arc (JoinNode, SplitNode) to N
8. Else if both JoinNode and SplitNode are places
9. Create a new transition t and add it to N
10. add arcs (JoinNode, t), (t , SplitNode) to N
11. Else if both JoinNode and SplitNode are transitions
12. Create a new place p and add it to N
13. add arcs (JoinNode, p), (p , SplitNode) to N
14. NextBlanket $\leftarrow \text{NextBlanket} - \text{Siblings}$
15. Return (N, OS)

Figure 16: **InsertLatents**: an algorithm to introduce hidden tasks between two sets tasks.

Algorithm **AddOption**
 Input N, workflow net;
 OptionalSet, an optional Set;
 Output N, workflow net

1. For each element x in OptionalSet
2. For each element y in x_3
3. create a transition t and add it to N
4. add arcs (x_1 , t), (t , y_1) to N
5. $x_3 \leftarrow x_3 - \{y\}$
6. If $\tau \in x_2$
7. create a transition t and add it to N
8. add arcs (x_1 , t), (t , z) to N where $\text{postSet}(z)$ is empty
9. Return N

Figure 17: **AddOption**: an algorithm to add optional pattern.

oracle. The tasks without any ancestor will be identified as CurrentBlanket in Step 3 of Figure 14. The hidden relationship between them (i.e., AND-splits and OR-splits) will be discovered by algorithms **HiddenSplits** in Figure 18 and **SplitStep**. The main function of algorithm **SplitStep** is to insert a piece of Petri Net representing a required split node. The hidden states will be replaced by a piece of Petri nets given in Figure 4. At the same time, we identify the first tasks of the workflow log not included in CurrentBlanket when we get rid of the existing tasks in current workflow net. The description of algorithm **Optional** is not given for the space limitation. If there are such tasks, there exists an optional pattern: because there are paths from CurrentBlanket to those tasks and thus the tasks along the paths are optional.

Algorithm **HiddenJoins**
 Input N, a Petri Net;
 S, a set of nodes;
 O, an ordering oracle;
 Output N, a Petri Net
 NewJoin, Node

1. If S has only one element t
2. Return (N, t)
3. Let M be a undirected graph (V, E) where $V = S$ and $E = \{(t_1, t_2) \mid t_1, t_2 \in S \text{ and } O(t_1, t_2) \neq \text{exclusive}\}$
4. If M is disconnected
5. $M' \leftarrow M$
6. Create a new Place NewJoin and add it to N
7. For each component C of M'
8. if C has only one node C_0
9. Add arc (C_0 , NewJoin) to N
10. Else
11. (N, NextJoin) $\leftarrow \text{HiddenJoins}(N, C, O)$
12. if NextJoin is place
13. Create a new transition t and add it to N
14. add arcs (NextJoin, t), (t , NewJoin) to N
15. Else
16. add arc (NextJoin, NewJoin) to N
17. Else
18. $M' \leftarrow$ the complement of M
19. Create new transition NewJoin and add it to N
20. For each component C of M'
21. if C has only one node C_0
22. Create new place p and add it to N
23. Add arc (C_0 , p) and (p , NewJoin) to N
24. Else
25. (N, NextJoin) $\leftarrow \text{HiddenJoins}(N, C, O)$
26. if NextJoin is place
27. add arc (NextJoin, NewJoin) to N
28. Else
29. Create new place p and add it to N
30. add arcs (NextJoin, p), (p , NewJoin) to N
31. Return (N, NewJoin)

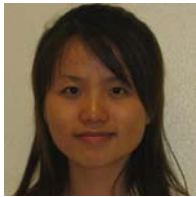
Figure 18: **HiddenSplits**: an algorithm for inserting required split nodes.

For each iteration (Steps 11-16), CurrentBlanket contains all and only the leaves of the current workflow net N. The next step is to find a set of tasks which will be added to N. The algorithm **GetNextBlanket** given in Figure 15 will find them. We then need to identify which tasks in NextBlanket are the descendants of the tasks in CurrentBlanket. This is done by the algorithm **Dependencies** not described in the paper due to the page limitation. The algorithm **Dependencies** returns as the result a DAG representing the ancestral relationship between the tasks in **CurrentBlanket** and in **NextBlanket**. The algorithm **InsertLatents** in Figure 16 will find hidden join/split tasks between CurrentBlanket and NextBlanket.

The main algorithm iterates until all observable tasks are added to the transition set of the workflow net. From the definition of workflow nets, there should be a final place to which there exists a path from every task. If the workflow net does not end with a single place, the algorithm **HiddenJoins** will complete the workflow net with a single place. The main function of algorithm **HiddenJoins** is to insert a piece of Petri Net representing a required join node.



Weixian Sun received B.S. degree in June 1997 from Sichuan University, China, and M.S. degrees in Computer science in June 2000 from Nanjing University, China. He studied for his Ph.D. degree in computer science since January 2002 at Florida International University. He joined Amazon.com in July 2006. He is working on Ph.D. dissertation as part-time. His research interest includes formal method, workflow mining, VOIP, formal modeling and Aspect-oriented software development.



Wei Peng received her B.S. degree in August 2002 from Xi'an Institute of Science and Technology, China. She is pursuing her Ph.D. degree in computer science from August 2003 with the honor of Presidential fellowship in Florida International University. Her research interests are data mining, machine learning, adaptive workflow management and bioinformatics.



Tao Li is currently an assistant professor in the School of Computing and Information Sciences at Florida International University. He received his Ph.D. degree in Computer Science from University of Rochester in 2004. His primary research interests are: data mining, machine learning, bioinformatics, and adaptive workflow management.



Dr. Tong Sun received her Ph.D. in Electrical and Computer Engineering from University of Rhode Island. She is a Principle Scientist in Xerox Research Lab Webster. Her research interests are: dynamic workflow modeling and management, semantic web, service orientated architecture and integration framework, knowledge engineering.