Mining Process Models with Duplicate Tasks from Workflow Logs

JiaFei Li, Dayou Liu*
College of Computer Science & Technology JiLin University, Changchun, 130012, China
Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education ,
Jilin University, Changchun 130012，China
*jiafei@jlu.edu.cn; dyliu@jlu.edu.cn(*corresponding author)*

Abstract

Current workflow management systems (WFMS) require user to provide explicit process models. The design of model is a difficult, costly and error-prone task. This presents a practical barrier to the adoption of workflow management technologies. A possible solution is process mining which can distill workflow models from a set of real executions. However, the present research in process mining still meets many challenges. The problem of duplicate tasks is one of them, which refers to the situation that the same task can appear multiple times in one workflow model. The α-algorithm is proved to mine sound Structured Workflow nets without task duplication. Combining techniques of machine leaning and the α-algorithm, a new algorithm called α that can deal with duplicate tasks is proposed and has been implemented in a research prototype. In nine scenarios, the α*-algorithm is evaluated experimentally to show its validity.*

Keywords: process mining, workflow mining, duplicate tasks, Petri nets, workflow nets.

1. Introduction

To produce more in less time, enterprises typically prescribe business processes that specify the way in which the resources are utilized. The quality and the accuracy of the business process directly decide the performance of an enterprise. Workflow management systems (WFMS) offer the functionality to manage and support operational processes.

Current workflow systems assume that a model of the process is available and the main task of the system is to insure that all the activities are performed in the right order and the process terminates successfully [3]. The user is required to provide the process model before it is enacted. Unfortunately, designing a formal model for an on-going, complex process is quite difficult, expensive, and error-prone. This forms a practical barrier to the adoption of workflow management systems.

To solve the problems mentioned above, instead of beginning with the process design, the technique of process mining starts by gathering information about the process execution and distracts a structured process description from these real executions. Process mining can ease the introduction of a workflow management system. An enterprise with an installed workflow system can also benefit from its help in the evaluation of the workflow system by comparing the distilled models with pre-defined models. It can also allow the evolution of the current process model into future versions of the model by incorporating feedback from successful process executions [3].

Data mining is the name given to the task of discovering information in data, which provide a stable foundation for process mining [10]. Different data mining methods can target different kind of data, such as relation database, images, time series and sequence data. Process mining handles the data which is the information recorded in the event logs and belongs to sequence data. Information systems using transactions (such as ERP, CRM and SCM) can provide such kind of data. The goal of process mining is to distill information about processes from event logs which record every event that occurred during workflow process execution. The event here refers to a task in a workflow instance and all events are totally ordered. The framework of process mining is depicted in Figure 1.

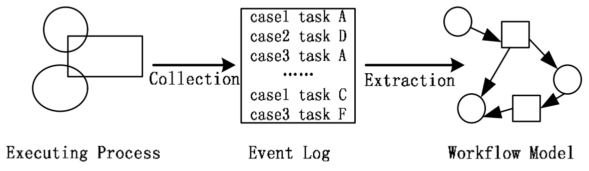

Figure 1.Framework of Process Mining

Process mining can be viewed as a three-phase process: pre-processing, processing and post-processing [8]. Most research in process mining focuses on mining heuristics primarily based on binary ordering relations of the events in a workflow log. A lot of work has been done on utilizing heuristics to distill a process model from event logs and many valuable progresses are made in the domain. However, all the existing heuristic-based mining algorithms have their limitations [8, 9]. There are still many challenging problems that the existing mining algorithms cannot handle. Duplicate tasks are one of them. It refers to the situation that one process model (e.g., a Petri net) has two or more nodes referring to the same task. Figure 2 shows a workflow model with three duplicate tasks (i.e. task X, task D and task E) represented in Petri nets. However, it is very difficult to automatically construct a process model from the event log of this model, because it is impossible to distinguish the task in one case from its cognominal task in the other cases.

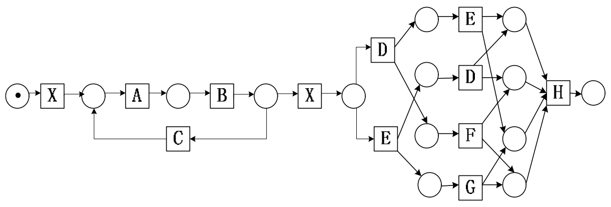

Figure 2. A workflow model with duplicate tasks

The α-algorithm [8] is proved to correctly distill sound Structured Workflow nets (SWF-nets, [8]) which have no task duplication [9]. The main idea of our method to handle the duplicate tasks is as follows. First, in the pre-processing phase, those tasks with same label are identified by our heuristic rules and marked with different labels in the log. Then, the α-algorithm is adopted to discover a workflow model from the identified log. Finally, during post-processing, the distilled model (in our case a Petri-net) is fine-tuned by recovering the marked task to their original label and a workflow model with duplicate tasks is obtained. The new mining algorithm based on the α-algorithm is name as α*.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the new approach to tackle task duplication using the α-algorithm. Section 4 concludes the paper and points out future work.

2. Related work

The idea of process mining is accepted widely for several years [3, 4, 5, 8, 10]. In the beginning, the research results are limited to sequential behavior. To extend to concurrent processes, Cook and Wolf propose several metrics (entropy, event type counts, periodicity, and causality) and apply them to distill models from event streams in [4]. However, they do not give any method to generate explicit process models. In [5, 6] Herbst and Karagiannis are also use an inductive approach to perform process mining in the context of workflow management. Two different workflow induction algorithms which are based on hidden Markov models are provided in [5]. The first method is a bottom-up, specific-to-general method and the other applies a top-down, general-to-specific strategy. These two strategies are limited to sequential models. The approach described in [6] is extended to tackle concurrency. Their approach is divided into two steps: induction step and transformation step. In the induction step task nodes are merged and split in order to extract the underlying process which is represented by stochastic task graphs. The stochastic task graph is transformed into an ADONIS workflow model in the transformation step. A notable difference with other approaches is that the approach allows for task duplication. The work of Aalst and his team members is characterized by the focus on workflow processes with concurrent behavior. In [10] a heuristic approach is provided to construct so-called “dependency/frequency tables” and “dependency/frequency graphs”. The approach is practical for being able to deal with noise. Another formal algorithm called α-algorithm is provided and proved to correctly distill workflow models represented in Petri-net from event logs and an extended version of the α-algorithm to incorporate short loops (i.e. length-one loops and length-two loops) is also presented in [8]. However, these algorithms are restricted to process models without duplicate tasks.

Compared with existing work, our work is characterized by the focus on concurrent workflow processes with task duplication behavior. Therefore, we want to distinguish duplicate tasks in the workflow log explicitly. To achieve this goal, the machine learning techniques are combined with Workflow nets (WF-nets, [2]) in this paper. Actually, WF-nets are a subset of Petri nets that provide a graphical but formal language to represent the workflow model. Our approach results in a workflow model of Petri-net directly without additional transformation step.

3. Solution to Tackle Duplicate Tasks

In this section the details of the new algorithm that can handle duplicate tasks are presented. First, the predecessor/successor table (P/S-table) of task which helps us to find duplicate tasks is constructed. Then, according to the P/S-table, several heuristic rules are given to identify the duplicate tasks. Last, an algorithm (called α*) that correctly mines sound WF-nets with duplicate tasks is provided.

3.1. Construction of the Predecessor/Successor Table

The starting point of our algorithm is to construct P/S-table of each task. For each task *A* that occurs in every workflow trace, the following information is abstracted out of the workflow log: (i) the name of the task that directly precedes task *A* (notation T_P), (ii) the name of the task that directly follows task *A* (notation T_S). The distilled information of task *A* is reserved in P/S-table.

Table 1. An event log of the model of Figure 2

case id	event trace
δ_1	*X A B X D F E H*
δ_2	*X A B X E D G H*
δ_3	*X A B X E G D H*
δ_4	*X A B X D E F H*
δ_5	*X A B C A B X D F E H*
δ_6	*X A B C A B X E D G H*
δ_7	*X A B C A B C A B X D E F H*
δ_8	*X A B C A B C A B X E G D H*

According to the process model of Figure 2, a random workflow log with 1000 event sequences (10550 event tokens) is generated. As an example, Table 1 shows the distinctive workflow traces which represent all the possible occurrences of every task in the log. The preceding task and the following task of every task *X* in each representative trace are listed in Table 2. The P/S-table seems clear without extra explanation except the notation of task identifier. The meaning of $\delta_i(t, N)$ is the *N*th occurrence of task named by *t* in the workflow trace called δ_i . For example, $\delta_1(X,1)$ is the first occurrence of task *X* in δ_1 and $\delta_5(X,2)$ is the second occurrence of task *X in* δ_5 . Notice that two nodes of task *X* both belong to the sequential event stream, while one node of task *D* is included in a concurrent event stream (the AND-split in *E*).

Table 2. An example P/S-table for task *X*

task identifier	T_P	T_S
$\delta_1(X,1)$	~	*A*
$\delta_1(X,2)$	*B*	*D*
$\delta_2(X,1)$	~	*A*
$\delta_2(X,2)$	*B*	*E*
$\delta_3(X,1)$	~	*A*
$\delta_3(X,2)$	*B*	*E*
$\delta_4(X,1)$	~	*A*
$\delta_4(X,2)$	*B*	*D*
$\delta_5(X,1)$	~	*A*
$\delta_5(X,2)$	*B*	*D*
$\delta_6(X,1)$	~	*A*
$\delta_6(X,2)$	*B*	*E*
$\delta_7(X,1)$	~	*A*
$\delta_7(X,2)$	*B*	*D*
$\delta_8(X,1)$	~	*A*
$\delta_8(X,2)$	*B*	*E*

Table 2 indicates that (i) the predecessors of $\delta_1(X,1)$ and $\delta_1(X,2)$ are different, (ii) the successors of $\delta_1(X,1)$ and $\delta_1(X,2)$ are also distinct, (iii) the predecessors and successors of $\delta_1(X,1)$ and $\delta_2(X,1)$ are identical, (iv) the predecessors of $\delta_1(X,2)$ and $\delta_2(X,2)$ are same while their successors are unlike. Finally, (v) if *X* is preceded by *B*, sometimes *X* is followed by *D* and sometimes by *E*.

Table 3. An example P/S-table for task *D*

task identifier	T_P	T_S
$\delta_1(D,1)$	*X*	*F*
$\delta_2(D,1)$	*E*	*G*
$\delta_3(D,1)$	*G*	*H*
$\delta_4(D,1)$	*X*	*E*
$\delta_5(D,1)$	*X*	*F*
$\delta_6(D,1)$	*E*	*G*
$\delta_7(D,1)$	*X*	*E*
$\delta_8(D,1)$	*G*	*H*

Table 3 depicts the predecessor and successor of task *D*. It can be concluded from Table 3 that (i) the predecessor and successor of $\delta_1(D,1)$ are quite different with those of $\delta_2(D,1)$ and $\delta_3(D,1)$, (ii) the predecessors of $\delta_1(D,1)$ and $\delta_4(D,1)$ are same while their successors are unlike, (iii) the predecessors and successors of $\delta_2(D,1)$ are cross-equivalent with those in $\delta_3(D,1)$ and $\delta_4(D,1)$. It is remarkable that the other occurrences of *X* and *D* in the left traces is similar with the above situations. In the next section we will use the P/S-table in combination with several relatively simple heuristics to identify the duplicate tasks.

3.2. Identification of Duplicate Tasks

The identification of duplicate tasks in a sequential workflow model is relatively easy. If it always the case that, the predecessors and successors of the tasks with same name are different, then it is plausible that they are two tasks owing same name. On the other hand, if the tasks sharing same name also have same predecessors and successors, it is no doubt that they refer to unique task. Nevertheless, the situation in a concurrent workflow model is more complicated. In many cases, although the cognominal tasks in two workflow traces have distinct predecessors and successors, we can not decide whether the two tasks are duplicate tasks or not, because the predecessors and successors may be cross-equivalent (i.e., the case that one predecessor equals to other successor). This occurs not only when the unique task belongs to a concurrent event stream but also when there are duplicate tasks.

In the previous section we observed that the information in the *X*-P/S-table strongly suggests that $\delta_1(X,1)$ and $\delta_1(X,2)$ are duplicate tasks because their predecessors and successors are quite different, and are not cross-equivalent also. Basing on the information in the in the *D*-P/S-table, the similar conclusion can be drawn on $\delta_1(D,1)$ and $\delta_2(D,1)$. In line with these observations, rule (1), the first heuristic rule to identify duplicate tasks is given below:

$$\text{IF}((T_P \neq T_P') \text{ AND } (T_S \neq T_S') \text{ AND } (T_P \neq T_S') \text{ AND } (T_S \neq T_P')) \quad (1)$$
$$\text{THEN} < \delta_i(t,N_1), \delta_j(t,N_2) > \in \text{U}$$

In rule (1), the first condition $(T_P \neq T_P')$ is used to judge that the predecessors of two occurrences of task *t* are different. The second condition determines the difference of their successors. Finally, the third condition and the fourth one state the requirement that there are not cross-equivalence between the preceding tasks and the following tasks. If four conditions are all satisfied, we can judge that the tuple consisting of two occurrences of task *t* belongs to U, the set of duplicate tasks. Applying this heuristic rule on the P/S-tables extracted from the log in Table 1, we obtain the result workflow log in Table 4. Comparing the workflow log of Table 4 and the process model of Figure 2, it can be seen that some of the duplicate tasks such as *X* is identified correctly. However, rule (1) can not distinguish the unique task belongs to a concurrent event stream with the duplicate tasks. For instance, $\delta_2(D,1)$ and $\delta_3(D,1)$ correspond to the unique task *D*, but they are marked as different tasks in Table 4.

In fact, in the case of cross-equivalence, if we can determine the task belongs to a concurrent event stream, the cognominal occurrences in two workflow traces can be confirmed to be a unique task, otherwise the two occurrences are corresponding to duplicate tasks. The property of the task in the concurrent case is illustrated by the following representative example. First, two functions of $pred(\delta,t)$ and $succ(\delta,t)$ are defined to get the predecessor and successors of task t in trace δ respectively.

Table 4. An identified event log of the log in Table 1

case id	event trace
δ_1	$X\ A\ B\ X_1\ D\ F\ E\ H$
δ_2	$X\ A\ B\ X_1\ E_1\ D_1\ G\ H$
δ_3	$X\ A\ B\ X_1\ E_1\ G\ D_2\ H$
δ_4	$X\ A\ B\ X_1 D\ E_2\ F\ H$
δ_5	$X\ A\ B\ C\ A\ B\ X_1\ D\ F\ E\ H$
δ_6	$X\ A\ B\ C\ A\ B\ X_1\ E_1 D_1 G\ H$
δ_7	$X\ A\ B\ C\ A\ B\ C\ A\ B\ X_1\ D\ E_2\ F\ H$
δ_8	$X\ A\ B\ C\ A\ B\ C\ A\ B\ X_1\ E_1\ G\ D_2\ H$

In event trace δ_3 of Table 1, the predecessor of $\delta_3(D,1)$ is G and $pred(\delta_3,G)$ is E which is just the predecessor of $\delta_2(D,1)$. In δ_2, the successor of $\delta_2(D,1)$ is G and $succ(\delta_2,G)$ is H which is just the successor of $\delta_3(D,1)$. The $\delta_1(E,1)$ and $\delta_4(E,1)$ also have the similar property. In line with the observations, the first heuristic rule (1) is extended with rule (2) and (3):

$$\text{IF}((T_P = T_S') \text{ AND } ((T_P' \neq pred(\delta_i,T_P)) \text{ OR } (T_S \neq succ(\delta_j,T_S'))) \quad (2)$$
$$\text{THEN} < \delta_i(t,N_1), \delta_j(t,N_2) > \in \text{U}$$
$$\text{IF}((T_S = T_P') \text{ AND } ((T_P \neq pred(\delta_j,T_P')) \text{ OR } (T_S' \neq succ(\delta_i,T_S))) \quad (3)$$
$$\text{THEN} < \delta_i(t,N_1), \delta_j(t,N_2) > \in \text{U}$$

Rule (2) and (3) specify the situation without concurrency. In rule (2), the first condition $(T_P = Ts')$ is used to judge that the predecessor of $\delta_i(t,N_1)$ and the successor of $\delta_j(t,N_2)$ are cross-equivalent. The second condition determines the difference of the predecessors of $\delta_j(t,N_2)$ and the predecessor of $\delta_i(t,N_1)$. And the third condition is similar with the second one which states the requirement that the successors of $\delta_i(t,N_1)$ and the successor of $\delta_j(t,N_2)$ are not equal. If these three conditions are all meet, we can determine that the tuple consisting of two occurrences of task *t* belongs to U. Rule 3 prescribes another similar situation. In the next section, the three rules are applied to identify the duplicate tasks in a workflow log.

3.3. Generating WF-nets from the identified workflow log

The solution to tackle duplicate tasks in sound WF-nets focuses on the pre- and post-processing phases of process mining [8]. The assumption about the completeness and noise free of a log is continued to use. The main idea is to identify the duplicate tasks and give them different identifiers. Any duplicate tasks can be identified by searching and checking the P/S-table of the task with the three heuristic rules above. The inspection of P/S-table follows the sequence of the occurrence of task in every event trace. If the task is determined to belong to duplicate tasks, it is renamed at the same time. The method of renaming is to append a serial number to the original task name.e.g.B1, B2, 1A, 1B. It is convenient that the original task name and the serial number are taken from distinct character sets. We need not to compare the task that has been checked and unmarked with its backwards cognominal tasks because the same task before it has compared with them already. Similarly, if the task to check has been marked, it is unnecessary to compare it with the original task. In Table 4, the task D_1 in δ_2 requires to compare with D_2 in δ_3, D_1 in δ_6 and D_2 in δ_8 instead of D in δ_4, δ_5 and δ_7.

The algorithm called α* based on these heuristics is presented in Figure 3. Let *T* be a set of tasks and *W* be a workflow log over *T*, the α-algorithm as in Definition 2.16 and the ordering relations as in Definition 2.14 in [8].

```
Algorithm  α*(W. N)
/*the extended α-algorithm to
     tackle duplicate tasks*/
1. $T_{log} \leftarrow \{t \in T \mid \exists_{\delta \in W}[t \in \delta]\}$.
2. $W^{-DT} \leftarrow W$.
3. isDup ← false.
4. isIdentify ← false.
5. FOR $\forall t \in T_{log}$ DO
        ( $\lambda \leftarrow buildPSTable(t, W^{-DT})$.
         FOR $\forall \theta \in \lambda$ DO
            ( $\lambda' \leftarrow \{\theta' \in \lambda \mid \theta' \neq \theta\}$.
              FOR $\forall \theta' \in \lambda'$ DO
       (isDup ← judgeDuplicate( $\theta, \theta'$ ).
        IF isDup THEN
          (// t' is the renamed task of task t
                    t' ← renameTask(t, $\theta', \lambda$ ).
                    $T_{log} \leftarrow T_{log} \cup \{t'\}$.
                    isIdentify ← true.).).).
         IF isIdentify THEN
            $W^{-DT} \leftarrow IdentifyLog(W^{-DT}, \lambda)$.).
6. $(P_{W^{-DT}}, T_{W^{-DT}}, F_{W^{-DT}}) \leftarrow \alpha(W^{-DT})$.
7. $P_W \leftarrow P_{W^{-DT}}$
8. $T_W \leftarrow eliminateTMark(T_{W^{-DT}})$.
9. $F_W \leftarrow eliminateFMark(F_{W^{-DT}})$.
10. $N \leftarrow (P_W, T_W, F_W)$ ▌
```

Figure 3. The α*-algorithm to mine duplicate tasks

The algorithm of α* works as follows. First, it examines the log traces (Step 1). Then the input log W^{-DT} to be processed by the α-algorithm, the flag *isDup* to describe whether there are duplicate tasks and the flag *isIdentify* to depict whether to identify the original input log W are initialized in steps 2 to 4. Then, in Step 5, the P/S- table λ of each task is generated, each table is checked to find the duplicate tasks based on the previous three heuristic rules in function *judgeDuplicate*, the found duplicate tasks are identified in tuple θ' of λ, the renamed task *t'* is added in T_{log} for further inspection and accordingly the previous log W^{-DT} is also marked and the result is still reserved in W^{-DT}. In Step 6, the α-algorithm discovers a workflow net based on the identified workflow log W^{-DT} and the ordering relations as defined in Definition 2.14 in [8]. The identifiers of the duplicated tasks are recovered to the original task name and their respective input and output arcs are adjusted accordingly in steps 7 to 9. Finally, the workflow net with the duplicate tasks is returned in Step 10. In the next section our experimental results of applying the above-defined approach on other workflow logs are reported.

3.4. Experiments

To evaluate the above described heuristics we have developed a research prototype which includes the α*-algorithm. The prototype can read a text file containing workflow traces produced by a WFMS of Staffware. By using Staffware, a wide variety of workflow traces of workflow models with different sizes and structural complexities can be generated. Nine different workflow traces are used to test our approach. One of these examples is taken from Herbst [5] to simply compare our method with model splitting. The number of tasks these models contain range from four tasks to twelve tasks which are shown in the working model of Figure 2 and the amount of duplicate tasks among them vary from one to four. Sequential processes, concurrent processes and loops are all included in our example models. For each model a random workflow logs with 1000 event sequences is generated.

Due to space limitation, it is not possible to depict all workflow models, the resulting P/S-tables and every WF-net deduced within the experiments in detail. Nevertheless, after applying our approach on the nine noise-free and complete workflow logs, the duplicate tasks in these logs are indeed identified accurately and the result WF-nets of the experiment is equivalent to the correct WF-nets. The equivalence here refers to structural equivalence. Mining the example log taken from Herbst [5] also results in an equivalent model with Herbst's result.

4. Conclusion and Future Work

In this paper, we focus on the extension of the α-algorithm so that it can mine WF-nets with duplicate tasks. The learning algorithm is named as α*. The α-algorithm is proven to correctly discover sound SWF-nets without task duplication. Changes in the pre- and post-processing phases are mainly involved in the extension. The details of α* is presented in three steps: Step (i) the construction of the P/S-table, step (ii) the identification of duplicate tasks based on P/S-table, and step (iii) the generation of the WF-net out of the identified workflow log using α-algorithm.

In the experimental section, we applied our algorithm on nine different workflow models with duplicate tasks. Sequential process, concurrent process and loops are included in these different models. For each model, we generated a random workflow log with 1000 event sequences. The experimental results show that our approach is valid to induce WF-nets with duplicate tasks.

In spite of the reported results, a lot of future work needs to be done. Firstly, the number of our test logs are limit, more experiments must to be done. Secondly, the theoretical basis of our learning algorithm needs to be improved. Finally, the above results are obtained by presupposing that the logs are complete and noise free. However, this situation appears rarely in real logs. Thus, to make our approach more practical, the heuristic mining techniques and tools which are less sensitive to noise and the incompleteness of log must to be developed.

Acknowledgement

This work is supported by NSFC Major Research Program 60496321, Basic Theory and Core Techniques of Non Canonical Knowledge; National Natural Science Foundation of China under Grant Nos. 60373098, 60573073, the National High-Tech Research and Development Plan of China under Grant No.2003AA118020, the Major Program of Science and Technology Development Plan of Jilin Province under Grant No. 20020303, the Science and Technology Development Plan of Jilin Province under Grant No. 20030523.

References

[1] W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, “Business Process Management: Models, Techniques, and Empirical Studies”, *Lecture Notes in Computer Science* , *Springer-Verlag*, Vol 1806, Berlin, 2000.

[2] W.M.P. van der Aalst, “The Application of Petri Nets to Workflow Management”, *The Journal of Circuits, Systems and Computers*, Vol 8, No.1, pp.21-66, 1998.

[3] R. Agrawal, D. Gunopulos, and F. Leymann, “Mining process models from workflow logs”, *Proceedings of the Sixth International Conference on Extending Database Technology*, pp.469-483, 1998.

[4] J.E. Cook and A.L. Wolf, “Event-Based Detection of Concurrency”, *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6)*, pp. 35-45, Orlando, FL, November 1998.

[5] J. Herbst and D. Karagiannis, “Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models”, *International Journal of Intelligent Systems in Accounting, Finance and Management*, Vol 9, pp.67–92, 2000.

[6] J. Herbst, “Dealing with Concurrency in Workflow Induction”, U. Baake, R. Zobel, and M. Al-Akaidi, editors, *European Concurrent Engineering Conference*, SCS Europe, 2000.

[7] S. Jablonski and C. Bussler, “Workflow Management: Modeling Concepts, Architecture, and Implementation”, *International Thomson Computer Press*, 1996.

[8] A.K.A de Medeiros, B.F. van Dongen, W.M.P. van der Aalst and A.J.M.M. Weijters, “Process Mining: Extending the *α*-algorithm to Mine Short Loops”, *BETA Working Paper Series*, *WP 113*, Eindhoven University of Technology, Eindhoven, 2004.

[9] A.K.A. de Medeiros, W.M.P. van der Aalst, and A.J.M.M. Weijters, “Workflow Mining: Current Status and Future Directions”, R. Meersman et al.,editors, *Lecture Notes in Computer Science*, Vol 2888, pp. 389-406, 2003,

[10] A.J.M.M. Weijters and W.M.P. van der Aalst, “Process Mining: Discovering Workflow Models from Event-Based Data”, B. Krose, M.de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pp.283–290, 2001.

JiaFei Li Doctoral student received the B.S. degree in Computer Software from Dept. Computer Science and Technology of Jilin University in 1997, and the M.S. degree in Computer Application Technology from Dept. Computer Science and Technology of Jilin University in 2000. Her research interests include: Workflow management technology, Process mining, Petri nets, Workflow nets and Data Mining.

DaYou Liu Professor. received the B.S. degree in Optics from Dept. Physics of Jilin University in 1966, and the M.S. degree in Computer Technology from Dept. Computer Science of Jilin University in 1981. His research interests include: Knowledge Engineering and Expert System, Multi Agent, Mobile Agents System and Intelligent Agent, Spatial-Temporal Knowledge Representation and Reasoning, Data Structure and Computing Algorithm, Rough Set and Data Mining, Data Structure and Computing Algorithm.