Process Mining: Extending α-Algorithm to Mine Duplicate Tasks in Process Logs

Jiafei Li, Dayou Liu, and Bo Yang

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, liudy, ybo}@jlu.edu.cn

Abstract. Process mining is a new technology 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. In this paper, basing on the "α-algorithm", a new algorithm (the "α*-algorithm") is presented to deal with duplicate tasks and has been implemented in a research prototype. In eight scenarios, the "α*-algorithm" is evaluated experimentally to show its validity.

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

1 Introduction

During the last decade workflow management concepts and technology have been applied in many enterprise information systems [1,7,10]. These systems usually require formal models of business processes to start the application. However, it is a time-consuming and error-prone task to acquire workflow models and adapt them to changing requirements, because knowledge about the whole process is usually distributed among employees and paper procedures. As indicated by many researchers, this requirement makes the workflow management systems inflexible and difficult to deal with change [1,10]. Process mining can be seen as a technology to contribute to this which aims at extracting a structured process description from a set of real executions recorded in a workflow log. And it can be viewed as a three-phase process: pre-processing, processing and post-processing [8].

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

K.C. Chang et al. (Eds.): APWeb/WAIM 2007 Ws, LNCS 4537, pp. 396–407, 2007.

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.

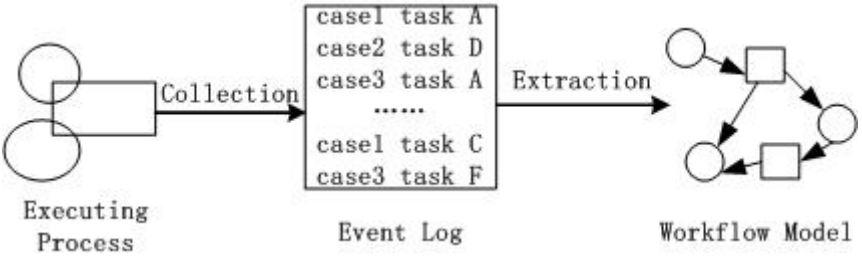

Fig. 1. Framework of Process Mining

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 the task owning the same name in the other cases.

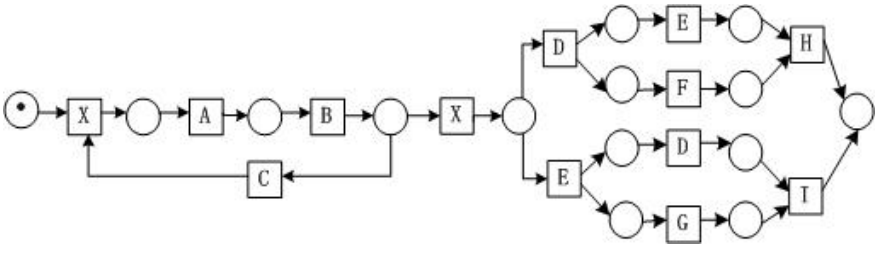

Fig. 2. A workflow model with duplicate tasks

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 Fig. 2

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

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 Nth 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*). To support the analysis of the information in P/S Table, we give the definition of cross-equivalent.

Definition 1 (cross-equivalent). Let "$\delta_i(t,N)$" and "$\delta_j(t,N')$" be two occurrences of task t in traces "δ_i" and "δ_j". T_P and T_S are predecessor and successor of "$\delta_i(t,N)$" in "δ_i", T_P' and T_S' are predecessor and successor of "$\delta_j(t,N')$" in "δ_j". If there is the situation that $T_P = T_S'$ or $T_P' = T_S$, then the situation is called cross-equivalent.

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 it is followed by *D* and sometimes by *E*. Table 3 shows the P/S table of task *D*.

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

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

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 "$\delta_4(D,1)$", (ii) the predecessors of "$\delta_1(D,1)$" and "$\delta_3(D,1)$" are same while their successors are unlike, (iii) the predecessors and successors of "$\delta_2(D,1)$" are cross-equivalent with those in "$\delta_1(D,1)$" 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 situations in a concurrent workflow model or choice workflow model are more complicated. In many cases, although the tasks owning the same name 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. This occurs not only when the unique task belongs to a concurrent event stream but also when there are duplicate tasks. Another possible case is that one task is both preceded and succeeded by a choice structure although there is no cross-equivalence. In this case, though the occurrences of task have different predecessors and successors in different traces, they are still corresponding to one unique task instead of duplicate tasks. Task E in Figure 3 shows the above situation. To avoid the situation that the unique task is determined as a duplicate task, we give the definition of selection relation.

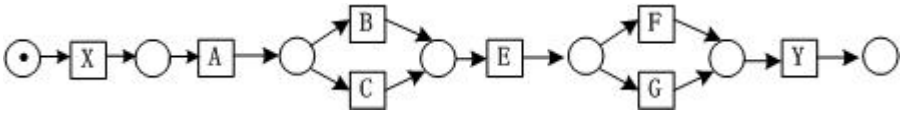

Fig. 3. Example Model of Selection Relation

Definition 2 (selection relation). Let T be the task set, W be a loop-complete workflow log over T. And let "$a,b \in T$", Task a and b have the selection relation if and only if there are traces "$\delta_1=t_1t_2t_3...t_n$", "$\delta_2=t_1't_2't_3'...t_n'$ " and "$i \in \{1,...,n\text{-}2\}$" such that "$t_i = a \wedge t_i' = b \wedge a \# b \wedge t_{i+1} = t_{i+1}' \wedge t_{i+2} = t_{i+2}'$ ". The notation of selection relation is "$\sum_W$".

Task B and C in Figure 3 have the selection relation, because in traces "δ_1=*XABEFY*" and "δ_2=*XACEFY*", we can find an integer $i=3$ which makes "$t_3 = B \wedge t_3' = C \wedge B \# C \wedge t_4 = t_4' \wedge t_5 = t_5'$ ".

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. Furthermore, their predecessors haven't the selection relation. Basing on the information in the in the *D*-P/S-table, the similar conclusion can be drawn on "$\delta_1(D,1)$"and "$\delta_4(D,1)$". In line with these observations, rule (1), the first heuristic rule to identify duplicate tasks is given below, let U be the set of duplicate tasks:

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

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. And the third condition and the fourth one state the requirement that there are not cross-equivalence between the preceding tasks and the following tasks. Finally, the fifth condition is to judge there is no selection relation between their predecessors. If five conditions are all satisfied, we can conclude that the tuple consisting of two occurrences of task *t* belongs to U, 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 identify all the duplicate tasks correctly. For instance, "$\delta_1(D,1)$" and "$\delta_2(D,1)$ " correspond to different task *D*, but they are not be marked separately in Table 4.

In fact, in the case of cross-equivalence, if we can determine the task belongs to a concurrent event stream, the 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"	$X\ A\ B\ X_1\ D\ E\ F\ H$
"δ_2"	$X\ A\ B\ X_1\ E\ D\ G\ I$
"δ_3"	$X\ A\ B\ X_1\ D\ F\ E\ H$
"δ_4"	$X\ A\ B\ X_1\ E\ G\ D_1\ I$
"δ_5"	$X\ A\ B\ C\ A\ B\ X_1\ D\ E\ F\ H$
"δ_6"	$X\ A\ B\ C\ A\ B\ X_1\ E\ D\ G\ I$
"δ_7"	$X\ A\ B\ C\ A\ B\ C\ A\ B\ X_1\ D\ F\ E\ H$
"δ_8"	$X\ A\ B\ C\ A\ B\ C\ A\ B\ X_1\ E\ G\ D_1\ I$

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

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

$$\text{IF } ((T_S = T_P') \text{ AND } ((T_P' \neq pred(\delta_j, T_P')) \text{ OR } (T_S' \neq succ(\delta_i, T_S))) \quad (3)$$

$$\text{THEN } <\delta_i(t,N_1), \delta_j(t,N_2)> \in U$$

Rule (2) and (3) specify the situation without concurrency. In rule (2), the first condition $(T_P = T_S')$ 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 predecessor of "$\delta_j(t,N_2)$" and the predecessor of T_P. And the third condition is similar with the second one which states the requirement that the successor of "$\delta_i(t,N_1)$" and the successor of T_S' are not equal. If these three conditions are all be met, 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. Table 5 gives the result after applying the three heuristics rules on the log in Table 4. It can be seen from Table 5 that all the duplicate tasks are identified correctly.

Table 5. An identified event log of the log in Table 4

case id	workflow trace
"δ_1"	$X\ A\ B\ X_1\ D\ E\ F\ H$
"δ_2"	$X\ A\ B\ X_1\ E\ D_2\ G\ I$
"δ_3"	$X\ A\ B\ X_1\ D\ F\ E\ H$
"δ_4"	$X\ A\ B\ X_1\ E\ G\ D_2\ I$
"δ_5"	$X\ A\ B\ C\ A\ B\ X_1\ D\ E\ F\ H$
"δ_6"	$X\ A\ B\ C\ A\ B\ X_1\ E\ D_2\ G\ I$
"δ_7"	$X\ A\ B\ C\ A\ B\ C\ A\ B\ X_1\ D\ F\ E\ H$
"δ_8"	$X\ A\ B\ C\ A\ B\ C\ A\ B\ X_1\ E\ G\ D_2\ I$

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. In fact, if the task to check has been renamed already, it is unnecessary to compare it with its forwards tasks owning the same name because the same task before it has compared with them already.

The algorithm called "α*" based on these heuristics is presented in Figure 4. 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].

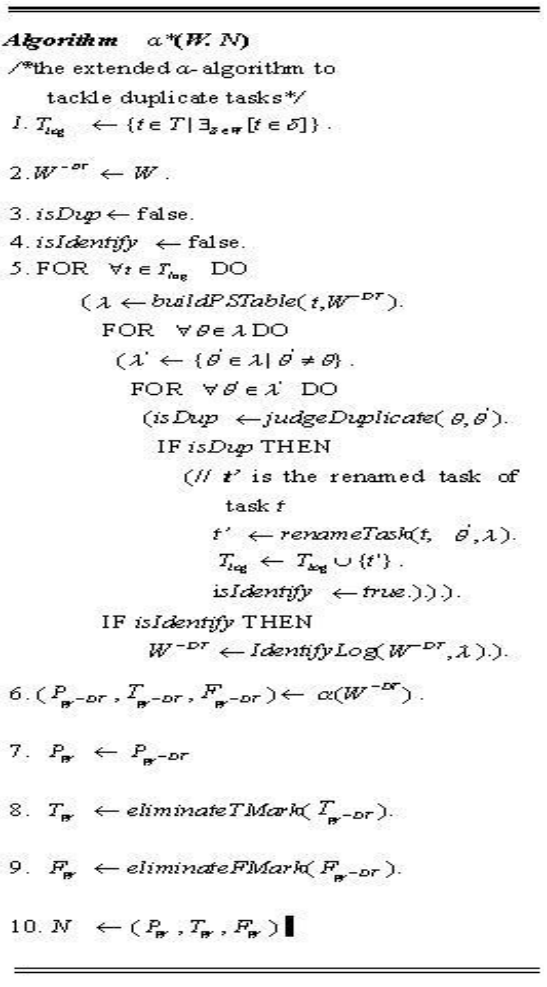

Fig. 4. The "α*-algorithm" to mine duplicate tasks

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. Eight 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. For each model a random workflow logs with 1000 event sequences is generated. In each log, the execution of tasks completely follows the sequence which is pre-defined in the WF-nets model. Then, we study the data in these logs under the experimental environment of P4(2.0GHz), 512 Ram and Windows 2000. The results of experiment are listed in Table 6 which is expressed in the metrics of event tokens number, duplicate tasks number, resultant model data (original model data) and mining time.

Table 6. Experiment Results

Model Name	Event Trace Number	Event Token Number	Duplicate Tasks Number	Resultant Model Data Transition Number/Place Number/Arc Number (Original Model Data)	Mining Time
M1	1000	5975	2	5/6/12 (5/6/12)	600ms
M2	1000	8750	1	5/6/11 (5/6/11)	670ms
M3	1000	2000	1	2/3/4 (1/2/2)	176ms
M4	1000	5000	1	5/7/12 (5/7/12)	687ms
M5	1000	6000	1	10/12/24 (10/12/24)	603ms
M6	1000	4000	1	5/5/10 (5/5/10)	562ms
M7	1000	8240	2	12/13/30 (12/13/30)	816ms
M8	1000	10550	3	12/14/33 (12/14/33)	1107ms

It can be learned from the experiment results in Table 6 that with the increase of the event token number in log, the time to mine model becomes longer gradually. And the more the duplicate tasks number in the WF-nets is, the longer the mining time becomes. The "α*-algorithm" can discover the example model (M1) adopted by Herbst correctly in less time and with higher efficiency. It is also shown in Table 6 that "α*-algorithm" can cost less time to recover most of the WF-nets to the original models and have stable performance. The reason that M3 can not be discovered precisely is the information included in the P/S-tables is few which makes our algorithm is unable to study correct knowledge from these limited data. In a word, the experiment results indicate that the "α*-algorithm" which identifies the duplicate tasks in logs and distills the identified logs later is valid and effective.

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 eight 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.

Acknowledgments. This work is supported by NSFC Major Research Program, Basic Theory and Core Techniques of Non Canonical Knowledge (No. 60496321); National Natural Science Foundation of China (No. 60373098, 60573073, 60503016), the National High-Tech Research and Development Plan of China (No. 2003AA118020), the Major Program of Science and Technology Development Plan of Jilin Province (No. 20020303), the Science and Technology Development Plan of Jilin Province (No. 20030523).

References

1. van der Aalst, W.M.P., Desel, J., Oberweis, A. (ed.): Business Process Management: Models, Techniques, and Empirical Studies. LNCS, vol. 1806. Springer, Heidelberg (2000)
2. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
3. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Proceedings of the Sixth International Conference on Extending Database Technology, pp. 469–483 (1998)
4. Cook, J.E., Wolf, A.L.: Event-Based Detection of Concurrency. In: Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6), pp. 35–45 (1998)
5. Herbst, J., Karagiannis, D.: Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. International Journal of Intelligent Systems in Accounting, Finance and Management 9, 67–92 (2000)

6. Herbst, J.: Dealing with Concurrency in Workflow Induction. European Concurrent Engineering Conference, SCS Europe (2000)
7. Jablonski, S., Bussler, C.: Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press (1996)
8. de Medeiros, A.K.A., van Dongen, B.F., van der Aalst, W.M.P., Weijters, A.J.M.M.: Process Mining: Extending the "α-algorithm" to Mine Short Loops. BETA Working Paper Series, WP 113, Eindhoven University of Technology, Eindhoven (2004)
9. 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., Tari, Z., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 389–406. Springer, Heidelberg (2003)
10. Weijters, A.J.M.M., van der Aalst, W.M.P.: Process Mining: Discovering Workflow Models from Event-Based Data. In: Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence, pp. 283–290. Springer, Heidelberg (2001)