Information Sciences 328 (2016) 558–576

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Process mining with token carried data

Chuanyi Li[a,b], Jidong Ge[a,b,*], Liguo Huang[c], Haiyang Hu[b,d], Budan Wu[b], Hongji Yang[e], Hao Hu[a], Bin Luo[a]

[a] State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, Nanjing 210093, China
[b] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
[c] Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275-0122, USA
[d] School of Computer, Hangzhou Dianzi University, Hangzhou 310018, China
[e] Centre for Creative Computing (CCC), Bath Spa University, England, UK

A R T I C L E I N F O

Article history:
Received 7 November 2014
Revised 20 June 2015
Accepted 31 August 2015
Available online 8 September 2015

Keywords:
Process mining
Process discovery
Workflow net
Petri net
Token
Token log

A B S T R A C T

Process mining is to discover, monitor and improve real processes by extracting the knowledge from logs which are available in today's information systems. The existing process mining algorithms are based on the event logs where only the executions of tasks are recorded. In order to reduce the pre-processing efforts and strengthen the mining ability of the existing process mining algorithms, we have proposed a novel perspective to employ the data carried by tokens recorded in token log which tracks the changes of process resources for process mining in this study. The feasibility of the token logs is proved and the results of pairwise *t*-tests show that there is no big difference between the efforts that are taken by the same workflow system to generate the token log and the event log. Besides, a process mining algorithm (τ) based on the new log is proposed in this paper. With algorithm τ, the mining efficiency as well as the mining capability is improved compared to the traditional event-log-based mining algorithms. We have also developed three plug-ins on top of the existing workflow engine, process modeling and mining platforms (YAWL, PIPE and ProM) for proving the feasibility of token log and realizing the token log generation and algorithm τ.

1. Introduction

When process mining was initiated, people were often curious about the difference and commonality between data mining and process mining. Data mining is aimed to extract or mine knowledge from a large corpus of data [7], while process mining targets at mining process models from data represented by process execution workflows. Therefore, data is the basis for both data mining and process mining. However, data is stored differently in these two cases. Data employed in data mining are often stored in databases, data warehouses, World Wide Web or other information repository [7], while data used in process mining are usually stored in logs of capturing the system workflows.

Logs used in traditional process mining research are called event logs [26]. Event logs are used to record and observe the process workflows from the control-flow perspective. Events represent execution of tasks. When the event logs are used to mine processes, not only three assumptions have to be satisfied [26] (see Section 2.3) but also a significant amount of efforts need to be taken in pre-processing the logs. To eliminate the three assumptions and reduce the pre-processing efforts in mining processes

[*] Corresponding author at: State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, Room B924, Feiyimin Building, #22 Hankou Road, Gulou District S, Nanjing 210093, China. Tel.: +86 13813968571.
E-mail address: gjdnju@163.com, gjd@nju.edu.cn, gjd@software.nju.edu.cn (J. Ge).

http://dx.doi.org/10.1016/j.ins.2015.08.050
0020-0255/© 2015 Elsevier Inc. All rights reserved.

on event logs, we here propose a new type of log named token log as the basis of process mining, which enables us to discover the process from a novel perspective of resources produced and/or consumed in the process. A token in Petri nets [15,17] is passed between transitions as the input or output of a transition. The token logs record the producer and consumer of tokens. The token logs can be directly acquired from the workflow management or other related systems just like the event logs. Section 3 defines and provides the details of the token logs.

Using token logs as the basis of process mining, we are able to save a significant amount of pre-processing effort for mining because we no longer need to distinguish different execution instances of a system and to generate the task traces, which have to be done in event-log-based process mining (see Section 2.3). The causal dependencies and parallel relationships among tasks can be directly observed from the token log. Besides, additional special structures could be discovered and the mining efficiency is improved as well. The major contributions we made in this paper are:

(1) We propose to use token logs, which can be easily generated from the information systems on the fly during the generation of event logs, as the basis of process mining. Extra data which have never been leveraged by the existing process mining methods are inserted into token logs to enhance both efficiency and ability of process mining.
(2) A novel process mining algorithm τ leveraging the data carried by token logs is proposed.
(3) Plug-ins for generating token logs and for implementing the mining algorithm are developed on top of the existing open source workflow engine and process mining tools.

Although one may argue that combining all the existing event-log-based mining algorithms (α [26], α^+ [4], α^{++} [29], β [30], λ [28]) may deliver the comparable ability for mining certain special structures as our token-log-based mining algorithm, Three major overheads resulted by combining multiple algorithms may compromise the mining efficiency:

(1) Different event logs are used by various mining algorithms. For example, β algorithm marks a type for each event and λ algorithm adds the post tasks into events, while there are no type or post tasks in the event log used by α algorithm. Hence the efforts for generating different kinds of event logs have to be taken into account.
(2) Each mining algorithm needs to be executed independently to visit all the entries in the event log for this algorithm. Combining the results of multiple mining algorithms requires multiple iterations of event logs for different algorithms, which duplicates the efforts of process mining.
(3) Models discovered by different mining algorithms may be different or even inconsistent. Then subsequent manual effort will be added to resolve the difference and inconsistencies.

Our token-log-based mining algorithm successfully avoids these overheads, which makes full use of the extra data recorded by the management systems on the fly with the generation of event logs and mines process models with the special structures correctly and more efficiently.

The rest of this paper is laid out as follows. Section 2 presents the background of process mining, Petri nets, WF-net and event log. Section 3 gives a detailed introduction to the token log, which leads the way to Sections 4 and 5. Sections 4 and 5 respectively describe the objectives of using token logs and the mining algorithm τ using token logs with a case study. Section 6 evaluates the τ algorithm and compares τ with α [26]. Section 7 discusses the limitation of applying τ. Section 8 summarizes the related work. Section 9 concludes the paper and envisages our future work.

2. Background

This section discusses the background of process mining. Section 2.1 gives an overview of process mining. Section 2.2 introduces the process modeling languages for the business processes used in this paper. Section 2.3 reviews the event logs.

2.1. Process mining

The term process mining is used for the method of distilling a structured process description from a set of real executions in [26]. Process mining is useful for at least in two aspects. First of all, it could be used as a tool to find out how people and/or procedures really work. Second, process mining could be used for delta analysis, i.e., comparing the actual process with some predefined process [26]. Based on the fact that (i) the advancements of multi-core and parallel technology resulted in a spectacular growth of the digital universe [2] (ii) the growth of a digital universe that is well-aligned with processes in organizations makes it possible to record and analyze more events [3], process mining has become one of hot topics in workflow technology.

Later in [22], the intension of term *process mining* is officially broadened to the idea of *process discovery*, *conformance checking* and *enhancement* [18,20,22]. Process discovery [21] takes an event log as input and produces a model without using any *a priori* information and this is the most prominent process mining technique. Conformance checking [18] aims to identify whether the reality, as recorded in the event logs, conforms to the model and vice versa. The idea of process enhancement is to extend or improve an existing process model leveraging information of the actual process execution recorded in event logs. This paper is focused on process discovery.

2.2. Petri nets and WF-net

Workflow nets (WF-nets) [25] are used in this paper for modeling workflow processes and the WF-net is a kind of Petri net. The graph in Fig. 1 is a WF-net and also a Petri-net.

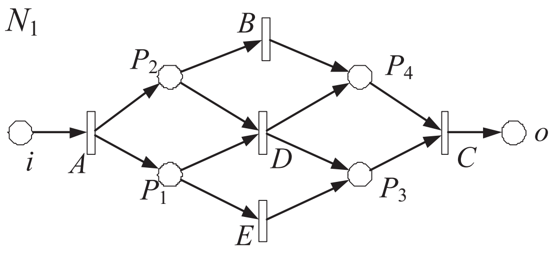

Fig. 1. An example of WF-net and Petri-net.

Petri nets are often denoted by a 3-tuple $PN = (P,T,F)$, where P is the set of places, T is a set of transitions and F is a set of arcs. All the pre-elements of an element $x \in P \cup T$ are in the set $\bullet x = \{y \in P \cup T | (y, x) \in F\}$ and the post-elements of x are in $x\bullet = \{y \in P \cup T | (x, y) \in F\}$. The element sequence $x_1, x_2,\ldots,x_n$ is called a *path* from x_1 to x_n while these elements satisfy (x_1, x_2), $(x_2, x_3),\ldots, (x_{n-1}, x_n) \in F$. Tokens are used in the Petri nets to simulate the dynamic and concurrent activities of systems [15]. Tokens are either placed in the places or consumed by the transitions. The presence of a token in a place indicates the holding of a true condition associated with the place. When k tokens are held in a place simultaneously, it indicates that k data or resources are available for post tasks of the place. The snapshot of the distribution of the tokens in all places of a net is called a marking or state, which is denoted by $M = (PN, M_0)$, where M_0 is the initial marking of PN. Transition t is enabled under marking M *iff* $\forall p \in \bullet t$: $M(p) \geq 1$, where $M(p)$ denotes the number of tokens in place p under M. All the enabled tasks under marking M build up the task set *enabled* (M). If a transition t is enabled, it will fire and the change of PN's marking is denoted as $M \xrightarrow{t} M'$. Below is the definition of *Reachable Marking* based on the transformation of marking:

Definition 1 (Reachable marking [15]). Let M_0 be the initial marking of Petri net $PN = (P,T,F)$. A marking M is *reachable* from M_0 *iff* there exists a path $\sigma = t_0, t_1, \ldots, t_n$ that satisfies $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} M_2 \ldots \xrightarrow{t_n} M$. The set of all reachable markings from M_0 is denoted by $[PN, M_0\rangle$.

In the remainder of the paper we assume that each process model is a sound WF-net. A WF-net requires the Petri net to have (i) a single *Start* place (i.e., input place i), (ii) a single *End* place (i.e., output place o), and (iii) each node must be on one path from *Start* to *End* [18]. The *soundness* property further enforces that (iv) there is no dead task, and (v) the process with only one token in the *Start* place can always terminate with only one token in the *End* place. If a WF-net is *sound* then there will be no dead-lock or live-lock in the model [18]. The initial and final states of a WF-net with input place i and output place o are denoted by $[i]$ and $[o]$.

Section 4.3 introduces the important definitions of WF-nets, which motivate the employment of token logs for process mining in details.

2.3. Event log

Process mining was originated to use system logs. Defining and unifying a system log becomes a key step for process mining. Different mining algorithms may use different kinds of event logs that could contain slightly different information as mentioned in Section 1.

In the event logs used by α algorithm [26], event is denoted by the item (*Case ID*, *Task Name*), where *Case ID* is used for marking different execution instances of workflow systems and *Task Name* represents the current executing task of the event. Here is an example event log for the process model N_1 in Fig. 1 described by a set of events: $Log_1 = \{$(case 1,A), (case 2,A), (case 3,A), (case 3,B), (case 1,B), (case 1,E), (case 2,E), (case 4,A), (case 2,B), (case 2,C), (case 5,A), (case 4,E), (case 1,C), (case 3,E), (case 3,C), (case 4,B), (case 5,D), (case 5,C), (case 4,C)$\}$. There are three key assumptions [26] for this kind of event logs: (i) each event refers to a single task; (ii) each event refers to a case (a workflow instance); and (iii) events are completely ordered (i.e., Events are sequentially ordered in the logs even if tasks may be executed concurrently).

To the event logs used in α algorithm, a large amount of pre-processing efforts are required. Firstly, we need to distinguish different *cases* for generating task traces for all *cases*. For example, the task traces in Log_1 are *ABEC* for cases 1 and 3, *AEBC* for cases 2 and 4, and *ADC* for case 5. Then we need to recognize all the ordering relations $>w$ [26] between tasks according to task traces (i.e., one task directly following the other in the task trace, e.g., $A > w\ B$ because of the trace '*ABEC*' for case 1). Finally, the causal dependencies ($\rightarrow w$ [26]) and parallel relationships ($\backslash\backslash w$ [26]) are discovered by $>w$ relations.

In the event logs used by λ algorithm [28], the event is denoted by $TaskName^{Number}[PostTasks]$ where *TaskName* is the current executing task of the event, the *PostTasks* contains all the post tasks of the current executing task and *Number* is the times of the event occurs. For example, the event log of N_1 in Fig. 1 used in λ algorithm is represented by a multiset: $Log_2 = \{A^1[BE], B^1[C], E^1[C], A^1[D], D^1[C], C^2[]\}$. Although the causal dependencies and parallel relationships can also be derived directly from the event log used in λ algorithm, the pre-processing effort for counting the times (field *Number*) that each event occurs still needs to be taken into account. Besides, an extra step is required to differentiate the *and-join* from the *or-join* relation based on the time of occurrence of each item. For instance, causal dependencies (B, C), (E, C) and (D, C) can be derived from the events

$e_1 = B^1[C]$, $e_2 = E^1[C]$, $e_3 = D^1[C]$ and $e_4 = C^2[]$ in Log_2. However, whether these dependencies are contained in an *and-join* or *or-join* relations cannot be determined. Although the parallel relationship between B and E is implied by the event $A^1[BE]$, it does not tell us whether there exists a parallel relationship between B and D, or between E and D. In this circumstance, such decision can be made based on the times of occurrence of events. Because $m(e_1) + m(e_3) = m(e_4)$ and $m(e_2) + m(e_3) = m(e_4)$ (the function m [28] returns the time of occurrence of the event e_i), λ-algorithm can determine it is the *or-joins* instead of *and-joins* between (B, D, C) and (E, D, C).

To eliminate the pre-processing efforts in generating task traces as well as extra steps for differentiating the *and-join* from the *or-join* when event logs are used for process mining, we propose that process mining can be done more effectively and efficiently based on token logs where additional data are leveraged as described in the next section.

3. Token log

3.1. Motivation and feasibility of token log

Tokens are created in Petri nets to represent various kinds of resources that are produced and/or consumed in a process. We aim to leverage data carried by tokens in process mining to track the state changes and footprints of resource usage in real world systems. The interesting questions are (1) why and how token logs can be leveraged for process mining; and (2) whether it is feasible to generate token logs without extra effort.

Motivation. There are many kinds of pre-conditions of tasks in business processes that provide various types of resources to trigger the task execution, such as timing, personnel and other resources. We reckon that if these conditions could be recorded in the logs with their producers and consumers, then the causal dependencies between tasks would be easily captured. Although there are many kinds of conditions, it would be ideal to use a unified entity in the process model to represent all these conditions to mine the dependencies among tasks. Token used in Petri-nets is a proper candidate. Various definitions and usage of tokens in Petri nets for process mining have been presented in previous studies (e.g., [18, 19]). A token is produced by a transition and consumed by its post transition and there is a causal dependency between these two transitions. If the token in the process models could have been utilized and augmented with extra data, we would be able to mine the task dependencies via the augmented tokens. For example, when a token is produced the name of its producer can be inserted into the token and when it is consumed the name of the consumer can also be added. Then the causal dependency between the token's producer and consumer can be directly derived from the token.

Feasibility. It is widely accepted that data in the event logs are already in the workflow systems while the systems are running. A natural question to ask is whether additional effort is incurred to generate token logs as compared to event logs. Fortunately, similar to event logs, data in the token logs are also readily available in real world workflow systems. Therefore, no extra effort is needed to generate the data for token logs. The data in token logs and those in event logs are simply collected on the fly from different perspectives of the running workflow systems. We have embedded a plug-in for making token logs in the workflow engine YAWL (*Yet Another Workflow Language*[1]) to show the feasibility of generating token logs and Fig. 3 shows the working theory of this plug-in.

As shown in Fig. 2, in order to develop the plug-in in YAWL, we mainly modify three classes and they are *YNetRunner*, *YTask* and *YCondition*. The *YEngine* class is used to create new *YNetRunner* objects for managing each newly started case. The method *initToken*() is added in the class *YNetRunner* for initializing tokens which would be used in the new cases. The methods *comsume*(), *print*() and *produce*() are the main methods added in class *YTask* for making token logs. Method *consume*() is used to consume some tokens from the input *YCondition* objects while a *YTask* object is ready to start and the consumers of the input tokens are set. Method *print*() will record the information of the tokens in logs after they are consumed. Before exiting its execution, the *YTask* object will create enough new tokens and put them into its output *YConditions* after setting their producers through method *produce*(). In class *YCondition*, a new member variety, list of tokens, and its setter and getter are added for managing the tokens passing it. With all these changes in YAWL engine, we can derive the token log which is defined in Section 3.2.

Section 6.1.2 compares the time taken to generate the event log and token log respectively and demonstrates that no additional effort is incurred to generate the token log.

3.2. Data carried by token

The log as the basis of mining shall contain sufficient amount of information to differentiate the three basic structures in processes and they are *sequence*, *selection* and *concurrent*. *Sequence* means two tasks are connected through one place directly (i.e., Fig. 3(a)) and this indicates causal dependency between these two tasks. A *selection* contains a place with one input task and two or more output tasks (i.e., Fig. 3(b)), or one or more input task and one output task (i.e., Fig. 3(c)). The special of a *concurrent* structure is the task with two or more output places (i.e., Fig. 3(d)) or input places (i.e., Fig. 3(e)).

Leveraging the data carried by tokens, *sequence* structures can be immediately captured by analyzing causal dependencies between tasks. In the *selection* and *concurrent* structures, the tasks which produce and consume tokens (resources) are identical so that simply relying on information of the tasks that produce and consume the resources cannot differentiate these two process

[1] YAWL is described at and can be downloaded from http://www.yawlfoundation.org/.

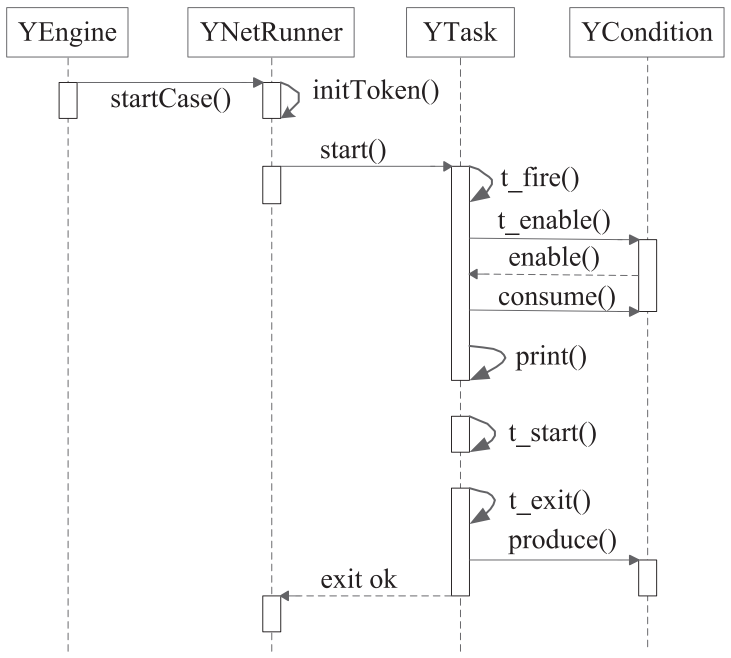

Fig. 2. The working theory of the plug-in for making token logs in YAWL.

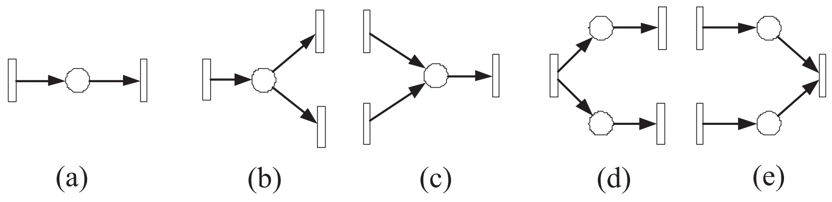

Fig. 3. Basic structures in processes: (a) sequence, (b and c) selection, (d and e) concurrent.

structures. However, we notice that the *selection* structure only produces one token while the *concurrent* structure produces the same number of tokens as the number of parallel branches. Therefore, if we can gather all the tokens produced by the input task in a *concurrent* structure during its single execution, then we can differentiate a *concurrent* structure from a *selection* one. Hence, we need a notation in token to uniquely identify in which execution of the producer the token is produced. We name the notation as Execution ID (*EID*) which uniquely identifies each execution of the task. We specifically define Execution ID as:

Definition 2 (Execution ID, *EID*)**.** Execution ID denoted by *EID*, is used to uniquely identify every execution of a task.

The token log shall not only record the names of resource producers and consumers but also the producers' and consumers' *EIDs* in order to be useful for mining the *sequence*, *selection* and *concurrent* structures. With the *EID* in the token log, we can tell that (1) there is at least a *sequence* structure between the producer and consumer of each token; (2) there is a *selection* structure between the producers and the consumers of tokens whose producers are the same but both producers' *EIDs* and consumers are different; (3) there is a *concurrent* structure between the producers and the consumers of tokens whose producers and producers' *EIDs* are both the same but consumers are different.

Table 1 is an example token log for N_1 in Fig. 1 and the token listed in the token log is defined as follows.

Definition 3 (Token)**.** A token is row in a token log and denoted by a quadruple $t = (PT, CT, PEID, CEID)$. *PT* is the producer of the token and *CT* is the consumer. *PEID* is the *EID* of *PT* which produces the token and *CEID* is the *EID* of *CT* which consumes the token. There are four operations that can be performed on a token: $PT = f_{pt}(t)$, $CT = f_{ct}(t)$, $PEID = f_{peid}(t)$, $CEID = f_{ceid}(t)$.

Note that some elements in the token quadruple (i.e., producer, consumer, producer *EID*, and consumer *EID*) can be *null* as shown in Table 1, which usually indicates the initialization and termination of a process execution. When a process is initialized, some tokens are inserted into the input place. When a process is terminated, the remaining tokens stay in the output place without being consumed. All the tokens in the log form a Token set that is defined as follows.

Definition 4 (Token set, *TS*)**.** Token set, $TS = \{tk_1, tk_2, \ldots, tk_n\}$, is the set of all tokens in the token log.

Table 1
The token log of the process model N_1 in Fig. 1.

Producer	Consumer	Producer execution ID	Consumer execution ID
Null	A	Null	1
A	B	1	2
A	E	1	3
B	C	2	4
E	C	3	4
C	Null	4	Null
Null	A	Null	5
A	D	5	6
A	D	5	6
D	C	6	7
D	C	6	7
C	Null	7	Null

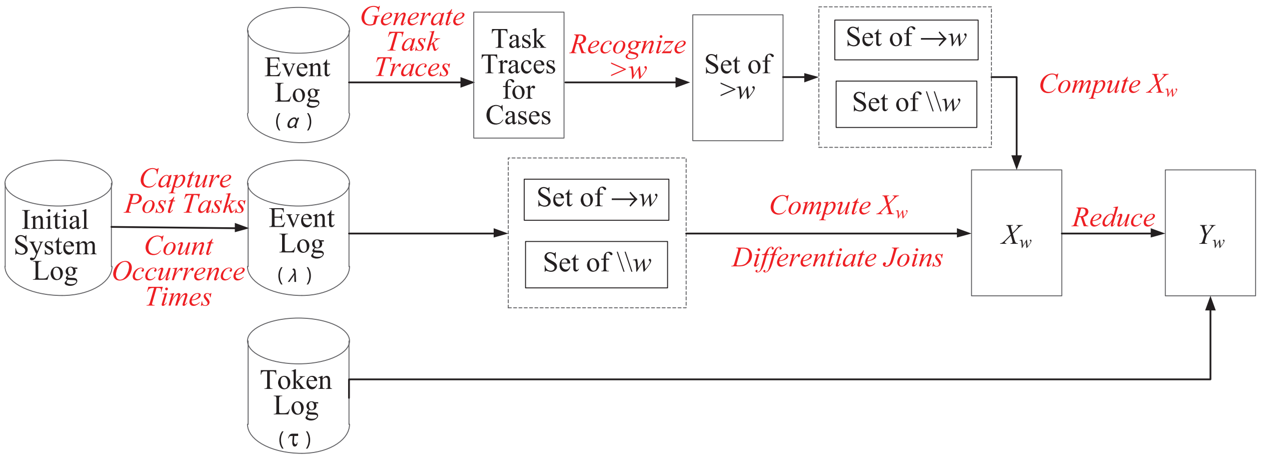

Fig. 4. Comparing the process mining workflows of α and λ algorithms on event logs with the mining workflow on token logs.

In the remaining sections, we will use *EID*, token and Token set (*TS*) to explain how data carried by tokens can be employed to enhance the effectiveness and efficiency of process mining.

4. Objectives

The three main objectives of leveraging the compressed token log in process mining is to simplify the process mining, improve the efficiency of mining algorithm by reducing the size of input logs, and enhance the capability of mining more special structures.

4.1. Objective 1: simplifying process mining

As analyzed in Section 2.3, when using event logs defined in α algorithm, causal dependencies and parallel relationships between tasks cannot be derived directly from the logs. Two pre-processing steps (i.e., *Generate Task Traces* and *Recognize* $>w$), as shown in Fig. 4, are necessary for discovering causal dependencies and parallel relationships. Another extra step is to compute the intermediate set X_W [26] (*Compute* X_W in Fig. 4) to generate the set of the pre and post task pairs for places (Y_W [26]) (*Reduce*).

λ algorithm adds the post tasks of the current executing task into the event logs. But the post tasks cannot be derived directly from the workflow management systems because the subsequent task to be executed cannot be determined when the current one is still running. Therefore, as shown in Fig. 4, the pre-processing efforts for capturing post tasks from initial system logs to generate events (*Capture Post Tasks*) and counting the times of occurrence of each event (*Count Occurrence Times*) are required. Besides, the step for differentiating *and-join* from *or-join* by the time of occurrence of events (*Differentiate Joins*) is also necessary (see Section 2.3).

Nevertheless, by leveraging data recorded in token logs, we simplify process mining by skipping the steps of *Generate Task Traces*, *Recognize* $>w$, *Compute* X_W, *Capture Post Tasks*, *Count Occurrence Times*, *Differentiate Joins* and *Reduce* as shown in Fig. 4. Section 5 will describe a novel process mining method leveraging data carried by token logs. Section 6.2 will evaluate the execution time between process mining based on event logs and token logs.

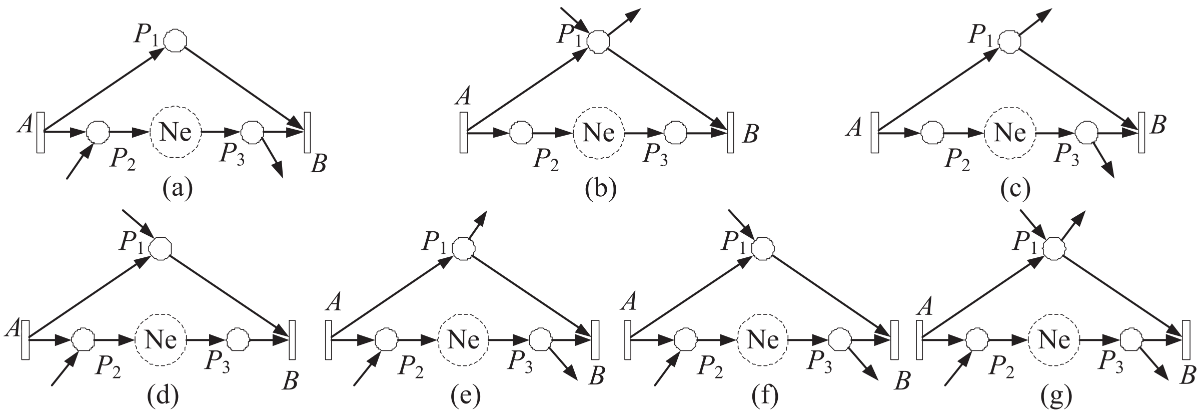

Fig. 5. Patterns of implicit dependencies that are described in [4].

4.2. Objective 2: reducing log size

Although it was mentioned in [28] that the size of the event log with post tasks was much smaller than that of the event log used in α algorithm, these event logs were actually generated from the initial system logs in much larger sizes. We reckon that to make a fair comparison of the log sizes, the initial system logs used to generate the event logs in λ algorithm shall be considered. Since it has been widely accepted that the event log used by α is the most basic event logs of workflow processes up to now, the initial system logs used by λ shall be generated from the event log used by α. Hence the sizes of logs used by λ cannot be smaller than the event logs used in α considering the sum of the log sizes of the its initial system logs (i.e., the event logs used by α) and the generated event logs (i.e., the event logs used by λ). In addition, since the event logs used in α^+, α^{++}, and β are extended from those used in α, their sizes must be larger than the log sizes of α. Therefore, we will only need to compare the size of the token log with that of event logs in α.

For the same process model, the token logs usually contain fewer entries than the event logs used by α algorithm. For example, the event log used by α algorithm (Log_1 in Section 2.3) of N_1 in Fig. 1 requires 11 entries while the token log (Table 1) has only 10 entries. Hence we hypothesize that the size of the token log is smaller than that of the event log for mining the same process model. Section 6.2 details the evaluation of the reduced log size.

4.3. Objective 3: handling more special structures

In order to describe the improvements made by our algorithm more understandable, we define four related process structures here first and they are *implicit dependency*, *implicit place*, *SWF-nets*, *one-loop-free workflow net*, and *two-loop-free workflow net*.

Definition 5 (Implicit dependency [29]). Let $PN = (P,T,F)$ be a sound WF-net with input place i and output place o. $\forall a,b \in T$, there is an implicit dependency between a and b *iff*: (1)$a\bullet \cup \bullet b \neq \emptyset$, (2) $\nexists M \in [PN, [i]\rangle \Rightarrow a \in enable(M) \wedge b \in enable(M - \bullet a + a\bullet)$, (3)$\exists M \in [PN, [i]\rangle \Rightarrow a \in enable(M)$, and $\exists M' \in [PN,M - \bullet a + a\bullet\rangle \Rightarrow b \in enable(M')$.

Fig. 5 shows the structure patterns of implicit dependency described in [4], where there is an implicit dependency between A and B. As described in [4,26,29–31], it is difficult to use event logs to mine these implicit dependency structures and it is not guaranteed that all the patterns can be discovered. But with token logs, we can easily discover all patterns of implicit dependencies.

Definition 6 (Implicit place [26]). Let $PN = (P,T,F)$ be a sound WF-net with an initial state $[i]$, $\forall p \in P$ is an implicit place *iff*: $\forall M \in [PN, [i]\rangle$, $\forall t \in p\bullet \Rightarrow$ *if* $M \geq \bullet t \setminus \{p\}$ then $M \geq \bullet t$.

The pattern of the implicit place is very similar to the implicit dependency. The major difference is that if the implicit place is removed from the process, the behavior of the process will not change.

Definition 7 (SWF-net [26]). A WF-net $PN = (P,T,F)$ with initial state M_i is a SWF-net (Structured Workflow Net) *iff*: (1)$\forall p \in P \wedge \forall t \in T \Rightarrow$ *if* $(p,t) \in F \wedge |p\bullet| > 1$ then $|\bullet t| = 1$, (2) $\forall p \in P \wedge \forall t \in T \Rightarrow$ *if* $(p,t) \in F \wedge |\bullet t| > 1$ then $|\bullet p| = 1$, (3) there are no implicit places (Definition 6).

Defining SWF-net here is used for illustrating the *non-SWF-net* (i.e., a WF-net contains one or more kinds of structures forbidden in SWF-net) which is used in describing the improvements made by our algorithm in mining ability. The patterns forbidden in SWF-net are shown in Fig. 6(a–c). Specifically, the second pattern (i.e., the parallelized input places of the same task have multi inputs) is named as *parallel places with multi inputs*.

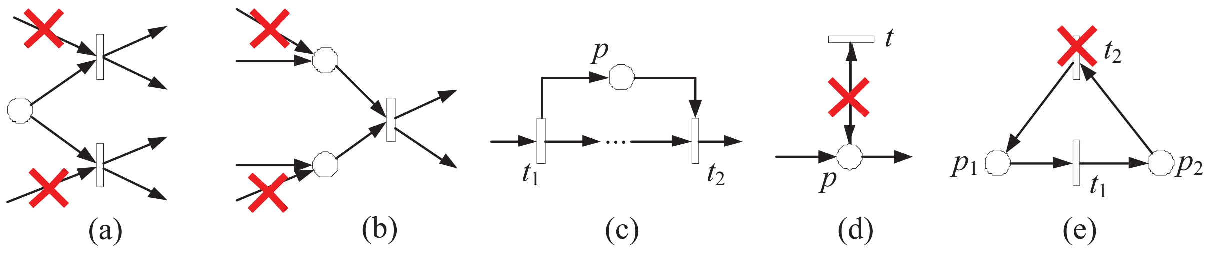

Fig. 6. (a and b) The first and second patterns forbidden in SWF-net; (c) pattern of implicit place; (d) structure forbidden in one-loop-free WF-net; (e) structure forbidden in two-loop-free WF-net.

Definition 8 (One-loop-free workflow net [4]). Let $PN = (P,T,F)$ be a workflow net. PN is a *one-loop-free* workflow net *iff* $\forall t \in T \Rightarrow t\bullet \cup \bullet t = \emptyset$.

Definition 9 (Two-loop-free workflow net). Let $PN = (P,T,F)$ be a workflow net. PN is an *two-loop-free* workflow net *iff* $\nexists t_1,t_2 \in T \Rightarrow t_1\bullet \cup \bullet t_2 \neq \emptyset \wedge t_2\bullet \cup \bullet t_1 \neq \emptyset$.

The structures defined in Definitions 8 and 9 are shown in Fig. 6(d and e) respectively.

All the existing mining algorithms have some restrictions in defining their mining ability. For instance, (1) α cannot mine implicit dependency, implicit places in non-SWF-nets, one-loop or two loop; (2) α^+ cannot mine implicit dependency or implicit places; (3) α^{++} cannot mine some of implicit dependencies, one-loop or two-loop; (4) β cannot mine implicit places in non-SWF-nets; and (5) λ cannot mine all implicit places or *parallel places with multi inputs* in non-SWF-nets. The algorithm τ proposed in this paper, which utilizes data carried by the token log, outperforms the existing algorithms in mining these four special structures. It has been explained in Section 1 that even if the combination of all the existing mining algorithms may be able to mine a majority of these special structures, our method using the token log still wins out by eliminating the three major overheads of running multiple mining algorithms. No wonder that none of the existing mining algorithms on event logs can mine all implicit places in non-SWF-nets.

So, the new mining algorithm τ is more effective than other event-log-based algorithms (i.e., α [26], α^+ [4], α^{++} [29], β [30], λ [28]) in mining (1) implicit dependency, (2) non-SWF-nets including *parallel places with multi inputs* and implicit places, (3) WF-nets with one-loop, and (4) WF-nets with two-loop. Section 5.3 will demonstrate τ algorithm's ability in mining all these special structures through a case study and a formal proof will be presented in Section 6.4.

5. Approaches

Existing process mining algorithms discover the pre and post tasks of all the places in the process model (Y_w) via mining the causal dependencies and parallel relationships among tasks from the event log and then reconstruct the entire process model. We here propose a novel algorithm (τ) that is able to mine the process model leveraging data carried by the token logs defined in Section 3. In this section, we first introduce the definitions and data structures based on which the τ algorithm is developed. Then we elaborate the pseudo code of the algorithm. Finally, we present a case study to demonstrate how the τ algorithm can be employed to mine special process structures.

5.1. Basic definitions and data structures

Causal dependencies and parallel relationships among tasks are defined differently in different kinds of logs. In event logs, causal dependencies and parallelism are defined based on the ordering relation $>w$ between tasks in task traces. In token logs, the causal dependency and parallelism are defined as follows.

Definition 10 (Causal dependency). Let $TS = \{t_1, t_2, \ldots, t_n\}$ be the token set of WF-net $PN = (P,T,F)$, and $a,b \in T$. The causal dependency between tasks is denoted by '$\rightarrow_w$' and $a \rightarrow {}_w b$ *iff*: $\exists t_i \in TS \Rightarrow a = f_{pt}(t_i) \wedge b = f_{ct}(t_i)$.

Definition 11 (Parallelism). Let $TS = \{t_1, t_2, \ldots, t_n\}$ be the token set of WF-net $PN = (P,T,F)$, and $a,b \in T$. The parallelism between tasks is denoted by '$||_w$' and $a||_w b$ *iff* $\exists t_i,t_j \in TS \Rightarrow [a = f_{ct}(t_i) \wedge b = f_{ct}(t_j) \wedge f_{pt}(t_i) = f_{pt}(t_j) \wedge f_{peid}(t_i) = f_{peid}(t_j)] \vee [a = f_{pt}(t_i) \wedge b = f_{pt}(t_j) \wedge f_{ct}(t_i) = f_{ct}(t_j) \wedge f_{ceid}(t_i) = f_{ceid}(t_j)]$.

As described in Section 3, each token entry includes the producer and consumer execution IDs (*EIDs*) of the token (resource) so that the causal dependency between the pre and post tasks of the place containing the token can be retrieved directly from the token log entry. We can then merge the token entries belonging to the same place to obtain the pre and post task sets of each place. In order to simplify the description of algorithm τ, we first define two data structures to describe the pre and post tasks of the place in Definitions 12 and 13.

Definition 12 (Pre–post contents of place, *PPCP*). Let $TS = \{t_1, t_2, \ldots, t_n\}$ be the token set of WF-net $PN = (P,T,F)$, and $p \in P$, then $PPCP = (PrTS, PoTS, PrEIDS, PoEIDS)$ is the pre–post contents of p, where the four elements are sets of pre tasks, post tasks, *EIDs* of

Table 2
Method *Merge*.

```
Merge: Merge a token into the PPCP
Input: Token $t^0 = (pt, ct, peid, ceid)$, $PPCP = (PrTS, PoTS, PrEIDS, PoEIDS)$
Output: $PPCP = (PrTS, PoTS, PrEIDS, PoEIDS)$
Begin:
    if $pt \in PrTS$
        if $ct \in PoTS$
            if $peid \notin PrEIDS$
            $PrEIDS$.add($peid$), $PoEIDS$.add($ceid$)
                Change status: $t^{0\to 1}$
            end if
    else
            if $peid \notin PrEIDS$
                $PoTS$.add($ct$), $PrEIDS$.add($peid$), $PoEIDS$.add($ceid$)
                Change status: $t^{0\to 1}$
            end if
    endif
  else
    if $ct \in PoTS$
            if $ceid \notin PoEIDS$
                $PrTS$.add($pt$), $PrEIDS$.add($peid$), $PoEIDS$.add($ceid$)
                Change status: $t^{0\to 1}$
            endif
        endif
    endif
End
```

pre and post tasks of p, respectively, iff: $\forall a \in PrTS \wedge b \in PoTS \Rightarrow \exists t_i \in TS \wedge f_{peid}(t_i) \in PrEIDS \wedge f_{ceid}(t_i) \in PoEIDS$. Then $p = p_{(PrTS,\, PoTS)}$. There are four operations on p for retrieving each element in *PPCP*: (1) $f_{prt}(p) = PrTS$; (2) $f_{pot}(p) = PoTS$; (3) $f_{preid}(p) = PrEIDS$; (4) $f_{poeid}(p) = PoEIDS$.

Definition 13 (*PPCP* set, *PSet*). $PSet = \{PPCP_0, PPCP_1, \ldots, PPCP_n\}$ is called the *PPCP* set of WF-net $PN = (P,T,F)$ *iff* $\forall p \in P \Rightarrow \exists PPCP = (PrTS, PoTS, PrEIDS, PoEIDS) \in PSet \wedge p = p_{(PrTS,\, PoTS)}$.

5.2. Mining algorithm

The main step of τ algorithm is to identify the tokens passing the same place during the executing of the process according to the relations between tokens. If tokens are passing a same place, then they shall be merged into the same *PPCP* (pre–post contents of place, a quadruple consisting of pre tasks, pre tasks' *EID*s, post tasks, and post tasks' *EID*s of a place as defined in Definition 12). We have developed an auxiliary method *Merge* in Table 2 to realize this in τ algorithm. The inputs of *Merge* are a token and a *PPCP* and the output is the *PPCP*. In *Merge*, the token's producer is added into *PrTS* (the pre task set of a place) of the *PPCP* based on the relation between the producer and *PrTS*. The token's consumer is added into *PoTS* (the post task set of a place) of the *PPCP* based on the relation between the consumer and *PoTS*.

With the definition of *Merge*, we can now describe the τ algorithm for mining the process model on the token log using the following pseudo code.

Definition 14 (Mining algorithm τ). The following pseudo code describes how the process model w can be mined based on the token log recorded from the executing system. T_w is the task set of the model. *TS* is the token set in the token log. *PSet* is the set of all *PPCP*s (pre–post contents of place) for the places in the model. P_w is the set of pre and post task pairs of all places and F_w is the set of all the arcs in the model.

```
1. $T_w = \{t_0, t_1, \ldots, t_n\}$
2. $TS = \{tk_0{}^0, tk_1{}^0, \ldots, tk_m{}^0\}$
3. Generate PSet from TS:
   for $i$ from 0 to $m$ do:
   (a) if $tk_i{}^{s\,==\,1}$ then continue, else do (b).
   (b) for PPCP in PSet, merge $tk_i{}^s$ into PPCP using method Merge;
       if $tk_i{}^{s\,==\,1}$ then continue, else do (c);
   (c) Build PPCP with $tk_i{}^s$: merge $tk_i{}^s$ into PPCP using method Merge; continue.
4. Form $P_w$ and $F_w$ with $PSet = \{p_1, p_2, \ldots, p_k\}$:
   $P_w = \{p_{(PrTS,\, PoTS)} | \exists j \in (0, k) \Rightarrow PrTS = f_{prt}(pj) \wedge PoTS = f_{pot}(pj)\}$
   $F_w = \{(t,p) | \forall p \in P_w, t \in f_{prt}(p)\} \cup \{(p,t) | \forall p \in P_w, t \in f_{pot}(p)\}$
5. Form the model: $w = (P_w, T_w, F_w)$
```

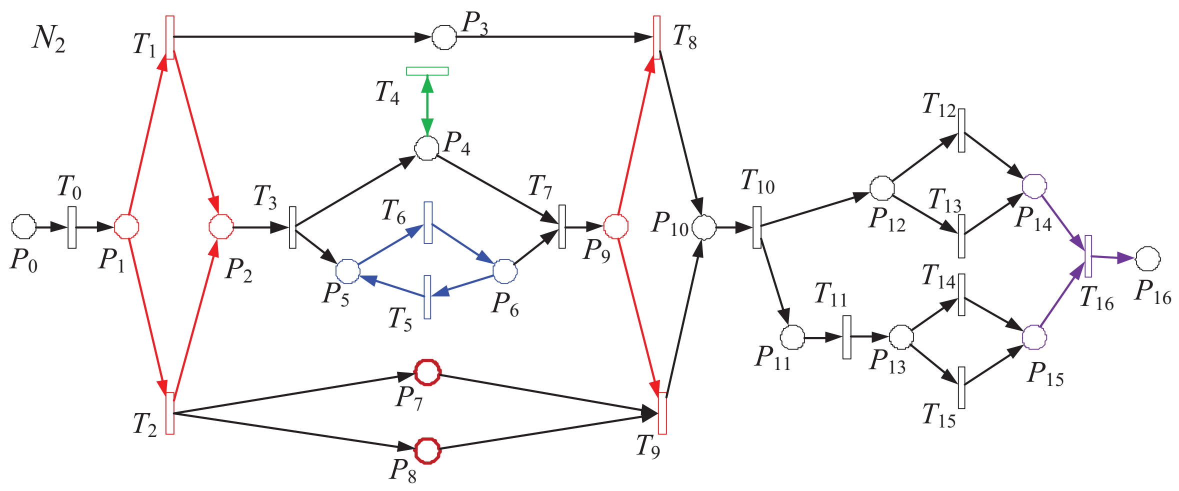

Fig. 7. A process model with the special structures (in colors) that cannot be completely discovered by the existing event-log-based process mining algorithms but can all be mined by token-log-based τ algorithm (red: implicit dependency; green: one-loop; blue: two-loop; deep-red: implicit place; purple: parallel places with multi inputs). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Next, we will demonstrate how our τ algorithm and the data carried by the token log can be used in tandem to mine a process model through a case study on a hypothetical process model.

5.3. A case study and experiments

We have experimented τ algorithm on the token log in Table 1 (i.e., the log of the process model N_1 in Fig. 1) to demonstrate its feasibility and effectiveness as well as to explain each step of applying τ. Through this case study, we would also like to demonstrate how the data carried by token log can be used at each step of process mining, for which we will show the change of the set of pre and post contents of places (*PSet*, which stands for the *PPCP* set in Definition 13) and the set of unhandled tokens (*UHT*) following each step of the algorithm. The following shows the steps of applying algorithm τ to Table 1 with the results of *PSet* and *UHT* bolded.

1. Form T_W, $T_W = \{A,B,C,D,E\}$;
2. Form *TS*, $TS = \{(,A,,1)^0,(A,B,1,2)^0,(A,E,1,3)^0,(B,C,2,4)^0,(E,C,3,4)^0,(C,,4,,)^0,(,A,,5)^0,(A,D,5,6)^0,(A,D,5,6)^0,(D,C,6,7)^0,(D,C,6,7)^0,(C,,7,,)^0\}$
3. Form *PSet*. The *UHT* is used. **Initially**: ***PSet* = Ø, *UHT* = *TS*.**
   3.1. Turn $tk = (,A,,1)^0$ to $PPCP = (\{\},\{A\},\{\},\{1\})$. Then $(,A,,5)^0$ is merged into *PPCP*. ***PSet* = {({},{A},{},{1,5})}. *UHT* = {(A,B,1,2)^0,(A,E,1,3)^0,(B,C,2,4)^0,(E,C,3,4)^0,(C,,4,,)^0,(A,D,5,6)^0,(A,D,5,6)^0, (D,C,6,7)^0,(D,C,6,7)^0,(C,,7,,)^0}.**
   3.2. No *PPCP* in *PSet* can merge $tk = (A,B,1,2)^0$ and turn it to $PPCP = (\{A\},\{B\},\{1\},\{2\})$ and merge remaining tokens in *TS*. ***PSet* = {({},{A},{},{1,5}),({A},{B,D},{1,5},{2,6})}. *UHT* = {(A,E,1,3)^0,(B,C,2,4)^0,(E, C,3,4)^0,(C,,4,,)^0,(A,D,5,6)^0,(D,C,6,7)^0, (D,C,6,7)^0,(C,,7,,)^0}.**
   3.3. Let $tk = (A,E,1,3)^0$, then merge *tk*. ***PSet* = {({},{A},{},{1,5}),({A},{B,D},{1,5},{2,6}),({A},{E,D},{1,5},{2, 6})}. *UHT* = {(B,C,2,4)^0,(E,C,3,4)^0,(C,,4,,)^0,(D,C,6,7)^0,(D,C,6,7)^0,(C,,7,,)^0}.**
   3.4. Let $tk = (B,C,2,4)^0$, then merge *tk*. ***PSet* = {({},{A},{},{1,5}),({A},{B,D},{1,5},{2,6}),({A},{E,D},{1,5}, {2,6}),({B,D},{C}{2,6}, {4,7})}. *UHT* = {(E,C,3,4)^0,(C,,4,,)^0,(D,C,6,7)^0,(C,,7,,)^0}.**
   3.5. Let $tk = (E,C,3,4)^0$, then merge *tk*. ***PSet* = {({},{A},{},{1,5}),({A},{B,D},{1,5},{2,6}),({A},{E,D},{1,5}, {2,6}),({B,D},{C},{2,6}, {4,7}),({E,D},{C},{2,6},{4,7})}. *UHT* = {(C,,4,,)^0,(C,,7,,)^0}.**
   3.6. Let $tk = (C,,4,)^0$, then merge *tk*. ***PSet* = {({},{A},{},{1,5}),({A},{B,D},{1,5},{2,6}),({A},{E,D},{1,5}, {2,6}),({B,D},{C},{2,6}, {4,7}),({E,D},{C},{2,6},{4,7}),({C},{},{4,7},{})}. *UHT* = Ø.**
4. Form P_W and F_W from *PSet*. $P_W = \{i = p_{(\{\},\{A\})}, P_2 = p_{(\{A\},\{B,D\})}, P_1 = p_{(\{A\},\{E,D\})}, P_4 = p_{(\{B,D\},\{C\})}, P_3 = p_{(\{E,D\},\{C\})}, o = p_{(\{C\},\{\})}\}$. $F_W = \{(i,A),(A,P_2),(A,P_1),(P_2,B),(P_2,D),(P_1,D),(P_1,E),(B,P_4),(D,P_4),(D,P_3),(E,P_3),(P_4, C),(P_3,C),(C,o)\}$.
5. Construct the process model *w*: $w = (P_W, T_W, F_W)$.

Further, we have conducted an experiment of τ on another hypothetical process model (N_2 in Fig. 7) containing various special structures that none of the existing process mining algorithms can completely discover. As mentioned earlier, α [26] cannot mine one-loop, two-loop and implicit dependency, α^+ [4] cannot mine implicit dependency, α^{++} [29], β [30] and λ [28] cannot mine non-SWF-nets. We have implemented τ algorithm by developing two plug-ins on the two open source tools for process modeling and mining, PIPE (*Platform Independent Petri Net Editor*, graphical tool for drawing and analyzing Petri nets[2]) and ProM (Process mining, the framework for process mining[3]), which will both be used in our experiment.

[2] PIPE can be downloaded from http://sourceforge.net/projects/pipe2/.

[3] ProM is described at http://www.processmining.org/ and can be downloaded from http://www.promtools.org/prom6/.

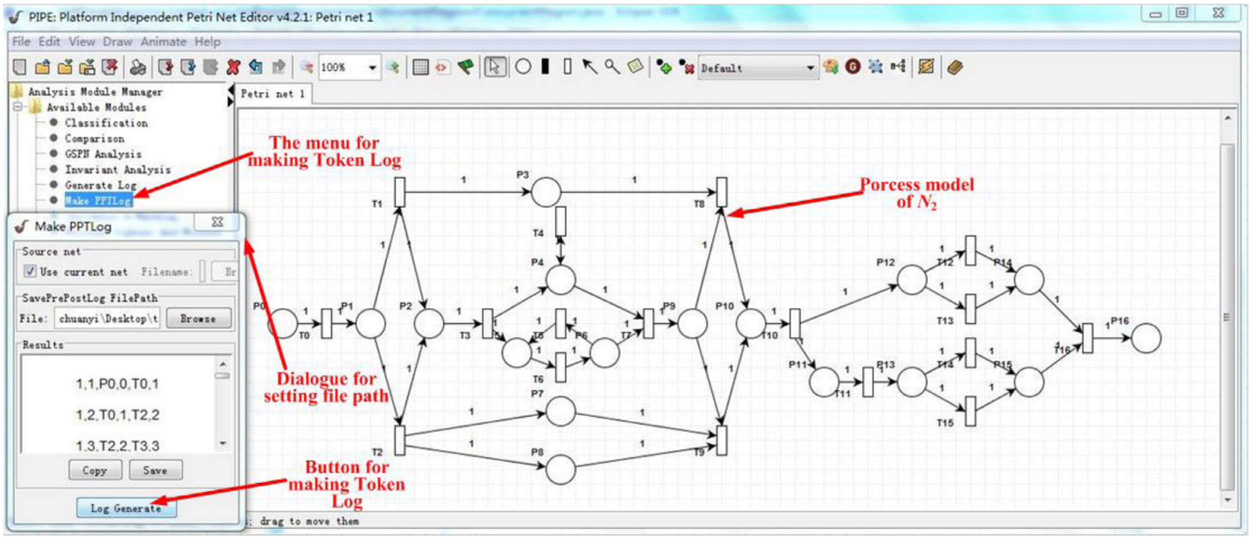

Fig. 8. Define N_2 and generate its token log in PIPE.

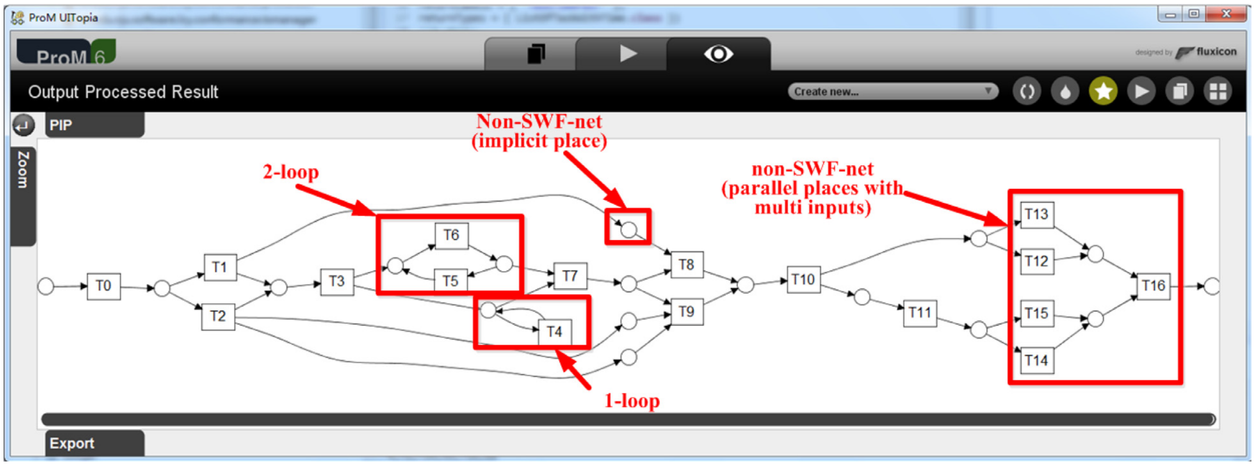
ProM UITopia
ProM 6
Output Processed Result
PIP
Zoom
Export
Non-SWF-net (implicit place)
2-loop
non-SWF-net (parallel places with multi inputs)
1-loop

Fig. 9. The process model mined by the TokenLog Mining Plug-in in ProM based on the token log of N_2.

Before explaining how to mine these four special structures, we first illustrate the steps of using the two plug-ins in PIPE and ProM in this experiment.

Step 1. Define the process N_2 in PIPE (see Fig. 8) and generate the token log by monitoring the execution of the process with the plug-in in PIPE.

Step 2. Import the token log captured by PIPE into ProM and choose the TokenLog Mining Plug-in in ProM for τ.

Step 3. Execute the TokenLog Mining Plug-in in ProM to obtain the mined process model (see Fig. 9).

The experiment result shows that implicit dependencies, non-SWF-net, and WF-nets containing one-loop and two-loop can now be correctly discovered by τ algorithm with the token log. Later, the mining ability of algorithm τ will be proved formally in Section 6.4.

Besides mining the hypothetical process model in Fig. 7, we have also applied the τ algorithm to mine the process model from the token log of a real world workflow system (i.e., *Patient Registration System* used in many hospitals in China as shown in Fig. 10) which also contains all of the aforementioned special structures. Our experiment results with the PIPE and ProM plug-ins show that all the special structures in the *Patient Registration System* can be correctly mined by the token-log-based τ algorithm.

6. Evaluations

6.1. Experiment setup and data collection

6.1.1. Experiment setup

In the following subsections, we will evaluate the τ algorithm on each of our three objectives to develop the new token-log-based process mining method as presented in Section 4. To evaluate the scalability and performance of the τ algorithm, we have

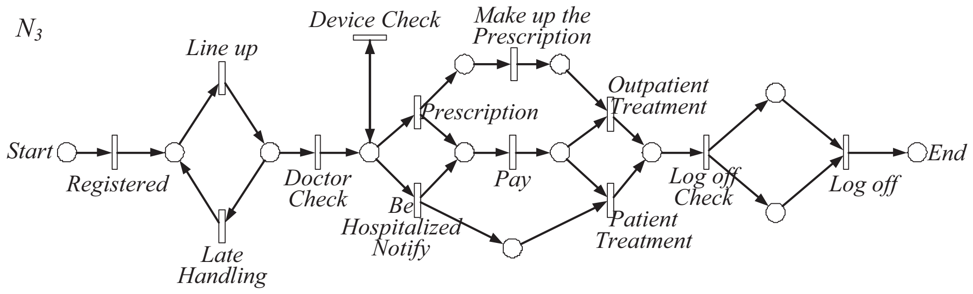

Fig. 10. *Patient Registration System* process used by many hospitals in China.

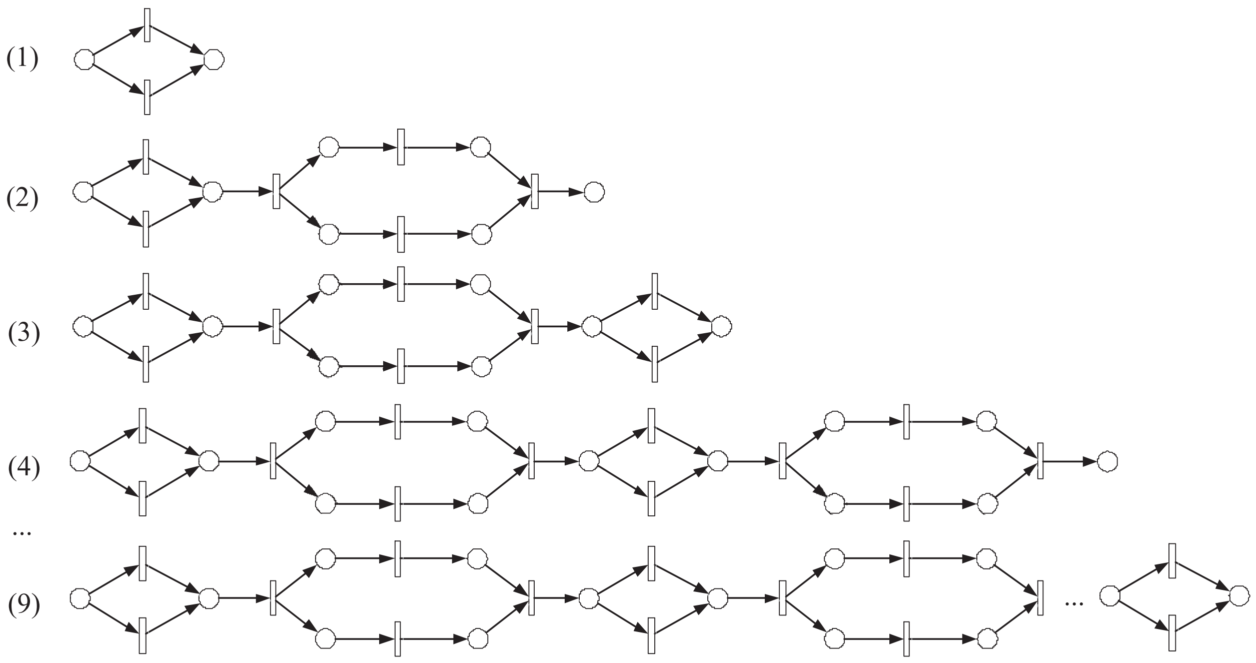

Fig. 11. Nine process models executed to compare the time of generating token logs and event logs.

conducted experiments to (1) compare the execution time between τ, α and λ; (2) compare the size of the token log with the event logs used by other algorithms. Finally, we will provide a formal proof that the four special structures can be completely mined by τ but not by the existing algorithms. Note that we have already presented our experiment results of mining the four structures in a hypothetical process model as well as in a real world patient registration process earlier in Section 5.3.

We conducted the experiments on the Intel(R) Core(TM) i3 computer with 4GB Memory and 32-bit Windows 7. PIPE 4.2.1 and ProM 6.3 were used in the experiments.

6.1.2. Data collection

We have described the approach of adding data carried by tokens into token logs for process mining in Section 3.1. Another interesting question is how easy it is to generate token logs as compared with traditional event logs. Next we will compare the time taken to generate the two kinds of logs for the same process models in Fig. 11 using the PIPE plug-in (see Section 5.3) which monitors the process execution.

Since the entries in the token log or event log are printed during task executions, it is difficult to measure the time only spent on printing logs. Therefore, instead, we measure the time taken to execute the entire process during which a token log or event log is printed respectively, as the log generation time. Each of the two logs is generated for ten times during the execution of each process model in Fig. 11 in order to eliminate the experimental errors and each log contains two running cases of the process. Table 3 lists the log generation times (*ns*).

We performed the pairwise *t*-tests on the log generation times for each of the 9 process models (see Table 3) and the null hypothesis is that times used for generating token logs are significantly different from those used for generating event logs. The results of the tests are shown in Table 4. It tells that the times taken to generate the token logs are a bit more than those used to generate the event logs, but we do not observe any statistical significance of the time differences since all the *P*-values are

Table 3
Time (ms) for generating the token log (TL) or event log (EL) from each of the 9 process models in Fig. 11.

Process models		Experiment no.									
		1	2	3	4	5	6	7	8	9	10
(1)	**TL**	15	15	16	16	16	16	16	16	16	15
	EL	15	16	15	16	16	15	16	16	16	16
(2)	**TL**	15	32	15	31	15	31	15	15	16	16
	EL	15	16	15	16	31	16	16	16	16	16
(3)	**TL**	15	15	15	15	31	15	15	31	15	15
	EL	31	15	15	16	16	15	16	16	16	15
(4)	**TL**	33	31	16	31	32	31	33	48	33	32
	EL	32	16	31	31	31	31	33	31	31	32
(5)	**TL**	16	33	32	33	31	32	31	48	32	31
	EL	31	32	33	31	33	32	32	32	32	16
(6)	**TL**	32	33	48	32	48	48	32	48	32	33
	EL	31	32	47	32	32	32	48	47	31	32
(7)	**TL**	48	32	48	48	33	47	47	32	48	32
	EL	31	32	32	47	46	48	48	47	32	48
(8)	**TL**	48	48	48	48	47	32	33	47	48	47
	EL	33	48	48	33	46	47	48	48	48	47
(9)	**TL**	48	32	32	48	48	32	48	48	48	48
	EL	48	32	48	48	32	47	48	32	47	48

Table 4
Results of pairwise *t*-tests applied to times taken to generate the token log (TL) and event log (EL) for each of the 9 process models in Fig. 11.

Pairs	Paired differences			*P*-values
	Mean	Standard deviation	Standard error mean	
(1) TL–EL	0.00000	0.66667	0.21082	1.000
(2) TL–EL	2.80000	9.91968	3.13688	0.395
(3) TL–EL	1.10000	8.79962	2.78268	0.702
(4) TL–EL	2.10000	8.82484	2.79066	0.471
(5) TL–EL	1.50000	8.78446	2.77789	0.602
(6) TL–EL	2.20000	8.97899	2.83941	0.458
(7) TL–EL	0.40000	12.70346	4.01719	0.923
(8) TL–EL	0.00000	10.01110	3.16579	1.000
(9) TL–EL	0.20000	10.50714	3.32265	0.953

far greater than 0.05. So the hypothesis is rejected which means there is no significant difference between the times used in generating token logs and those used in generating event logs.

The tokens logs and event logs generated here will be used in the experiments for comparing the executing time between algorithm τ and algorithms α and λ in Section 6.2.

6.2. Objective 1: simplifying the mining process—executing time

As shown in Fig. 4, leveraging token logs eases process mining by skipping several steps. In this subsection, we would like to show our experiment results of comparing the executing time of different algorithms (α, λ and τ) for mining the nine process models in Fig. 11. Considering that the event logs used by λ algorithm cannot be directly derived from the workflow systems, the times used for generating them are taken into account. Since the mining algorithms α^+, α^{++}, β, are extensions of α, which means that their efficiencies are not as good as α, so the comparison between these algorithms and τ is omitted.

The execution time of the algorithm on each of the nine process models is measured to cover the steps of the corresponding algorithm shown in Fig. 4. We repeat the mining work for each algorithm on each process model for 7 times in order to eliminate the experimental errors. Table 5 lists the experiment results. Fig. 12 illustrates the increasing average execution time of all three algorithms (α, λ and τ) with the increasing complexity of the process models. It shows that the execution time of τ is consistently lower than both α and λ. We perform the pairwise *t*-tests on the execution time between (α, τ) and (λ, τ) and the null hypotheses are: (1) algorithm α uses significantly more time than τ in handling models in Fig. 11, and (2) algorithm λ uses significantly more time than τ in handling models in Fig. 11. The results of *t*-tests are shown in Table 6.

According to Fig. 12, the average executing times used by α and λ are longer than used by τ in general. That all *Sig.(2-tailed)*s (i.e., *P*-values) of *t*-tests in Table 6 are less than 0.05 indicates the null hypotheses should not be rejected, which means executing times used by α and λ are really more than those used by τ in mining models. Taking the standard. deviations in Table 6 into consideration, it is concluded that the efficiency of τ is significantly higher than those of α and λ.

Table 5
Comparison of execution time (ns) of α, λ and τ in mining the 9 process models in Fig. 11.

Model		1	2	3	4	5	6	7
(1)	α	6158020	4237907	3575441	8164833	12221306	4805973	8348274
	τ	97746	144611	297926	148628	324705	151641	176747
	λ	340773	347132	329391	338763	575764	308302	317675
(2)	α	8789472	11131367	5753310	11252546	12466342	6130906	12080712
	τ	379270	333074	328723	536600	325375	342112	301608
	λ	1029346	1063156	1075876	1057800	1052444	952020	973779
(3)	α	10881645	10089967	8862781	12417134	11229448	8585276	10638953
	τ	624305	377596	530575	717365	369897	369897	677529
	λ	1356729	1632560	1080897	1289444	1008257	1100313	940304
(4)	α	10414672	9631697	10719627	18550046	12807114	9829533	12368930
	τ	502122	1063493	497435	486724	1120735	489736	666483
	λ	1877595	1747044	1433385	1424682	1514059	1436398	1868221
(5)	α	27594597	16671108	30529665	28741443	27830258	25952324	15253451
	τ	607902	1217143	497436	722051	1182999	511829	509821
	λ	1714573	1762441	1696162	1754073	1720264	1696162	1855167
(6)	α	44201767	32040381	45400834	42983620	36960169	34021419	36034591
	τ	1570302	690585	697280	805737	693597	662800	675521
	λ	2264227	2314104	2295023	2242804	2188574	2188909	2294020
(7)	α	56691875	69713229	57465813	73842677	69173280	54650249	56554963
	τ	880721	979138	713682	717365	858963	1236893	758873
	λ	2516292	2547088	2671949	2504575	2518969	2619393	2514952
(8)	α	120482077	122378758	136271459	126029182	128277682	132811506	138107550
	τ	1200405	994870	925243	888085	874361	1491971	880722
	λ	3746821	3380607	3604553	3515175	3492747	3540616	3522540
(9)	α	356666702	257555926	248228180	270033314	240740208	269970381	253683229
	τ	1149524	927920	935955	1210448	1209779	924238	1400250
	λ	3635684	3873355	3705981	4016626	3598193	3736109	3608235

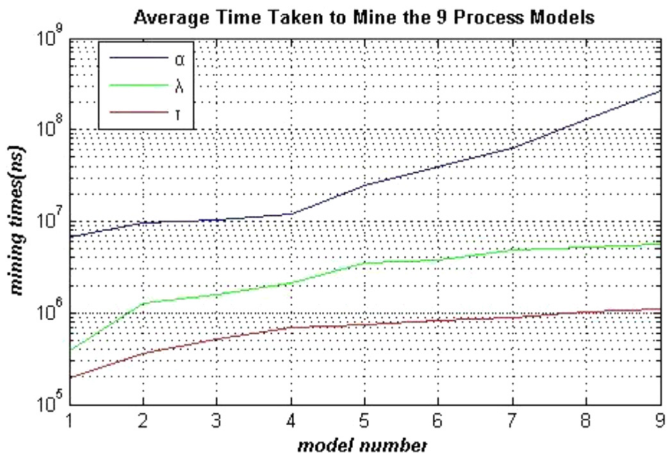

Fig. 12. Comparison of the average execution time of α, λ, and τ to mine each of the 9 process models in Fig. 11.

6.3. Objective 2: reducing log size

We have discussed in Section 4.2 that the size of event logs used by λ algorithm for process mining cannot be smaller than those used by α algorithm. Thus we only compare the size of token logs used by τ with that of event logs used by α. When calculating the size of the event logs, we will use the minimized event logs just sufficient for α to correctly mine the process models [26].

To a hypothetical process model consisting of n selections with k choices in each selection, there are $2k + (n-1)k^2$ tokens in the token logs and nk^2 events in the event logs. Fig. 13(a) plots the sizes of the token logs and event logs with the increasing number of selections and choices. To another hypothetical process model composed of n parallel structures with k branches in each parallelism, there are $2 + 2nk$ tokens in the token logs and $nk^2 + (n+1)k$ events in the event logs. Fig. 13(b) plots the sizes of the token logs and event logs with the increasing number of parallelisms and branches.

In general, as shown in Fig. 13, no matter how complex the process model is, the number of event entries in its event log is always greater than the number of token entries in its token log. Besides, the more selections/choices and parallelisms/branches the model contains, the faster the number of entries in its event log increases than the number of entries in its token log.

Table 6
Paired t-tests of algorithm execution time: (α, τ) and (λ, τ) on each of the 9 process models respectively.

Algorithm pairs	Paired differences		P-values
	Mean	Standard deviation	
(1) $\alpha-\tau$	6.6E6	2998015.5	0.001
$\lambda-\tau$	1.7E5	74706.684	0.001
(2) $\alpha-\tau$	9.3E6	2787044.9	0.000
$\lambda-\tau$	6.7E5	80440.506	0.000
(3) $\alpha-\tau$	9.9E6	1265747.3	0.000
$\lambda-\tau$	6.8E5	300027.17	0.001
(4) $\alpha-\tau$	1.1E7	3173412.7	0.000
$\lambda-\tau$	9.2E5	321777.32	0.000
(5) $\alpha-\tau$	2.4E7	6180566.3	0.000
$\lambda-\tau$	9.9E5	322917.49	0.000
(6) $\alpha-\tau$	3.8E7	5157298.7	0.000
$\lambda-\tau$	1.4E6	330753.52	0.000
(7) $\alpha-\tau$	6.2E7	8025476.1	0.000
$\lambda-\tau$	1.7E6	181717.56	0.000
(8) $\alpha-\tau$	1.3E8	6810022.3	0.000
$\lambda-\tau$	2.5E6	223994.92	0.000
(9) $\alpha-\tau$	2.7E8	39268548	0.000
$\lambda-\tau$	2.6E6	270914.31	0.000

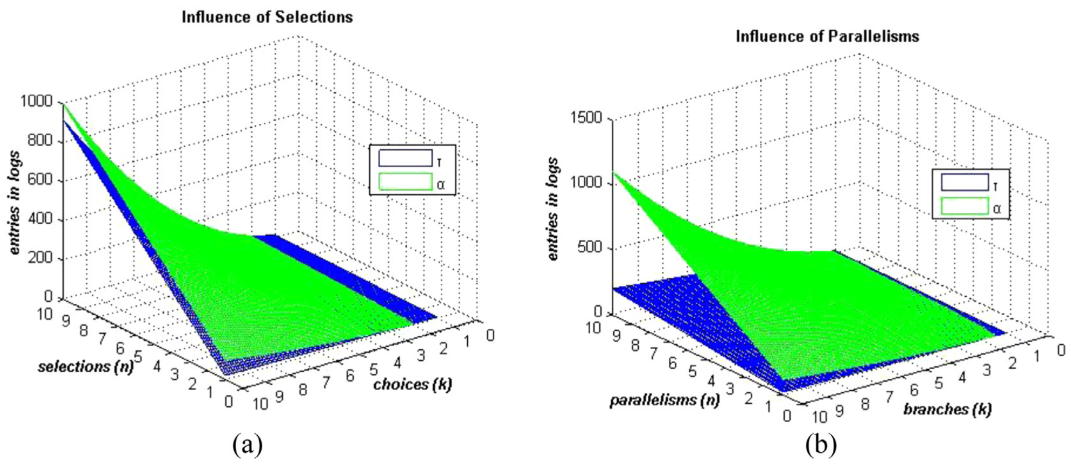

Fig. 13. (a) Comparing the sizes (number of entries) of event logs and token logs with the increasing number of selections and choices in a process model; (b) comparing the sizes of event logs and token logs with the increasing number of parallelisms and branches in a process model.

6.4. Objective 3: mining special process structures

Section 5.3 has already demonstrated that our token-log-based mining algorithm τ is able to discover more special process structures than other existing mining algorithms (e.g., α, α^{+}, α^{++}, β and λ) through the experiments on a hypothetical process model as well as a real world patient registration process for hospitals in China. This section proposes a new theorem and its proof to formally validate the ability of algorithm τ in mining the three special process structures.

Theorem 1. *The token log based algorithm τ is able to mine all four types of process structures including implicit dependency* (Fig. 5), *non-SWF-net* (parallel places with multi inputs (Fig. 14(a)) *and implicit place* (Fig. 6(c))), *one-loop* (Fig. 14(b)) *and two-loop* (Fig. 14(c)), *which cannot be fully discovered by any of the existing event log based mining algorithms including α, α^{+}, α^{++}, β and λ.*

Proof.

(1) Mine implicit dependency:

$\because$ In Fig. 5, (implicit dependency $\Leftrightarrow P_1$) $\wedge$ (Mine Place $\Leftrightarrow$ Make *PPCP* (pre–post contents of places))
$\therefore$ Mine implicit dependency between A and $B \Leftrightarrow$ Make the *PPCP* of P_1
$\because$ Ne between A and B (Ne is any task or tasks)
$\therefore$ Tokens produced by A or consumed by B are:

$$t = (A, EID_A, B, EID_B), t' = (A, EID_A, Ne, EID_{Ne}) \quad and\ t'' = (Ne, EID_{Ne}, B, EID_B)$$

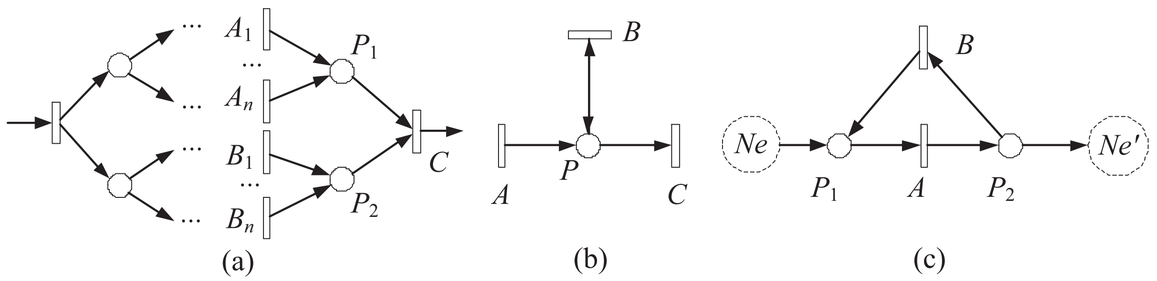

Fig. 14. (a) The general pattern of *parallel places with multi inputs* (contrast to Definition 6 (2), Fig. 6(b)); (b) pattern of one-loop; (c) pattern of two-loop.

$\because$ [$f_{pt}(t) = f_{pt}(t') \wedge f_{peid}(t) = f_{peid}(t')$] $\wedge$ [$f_{ct}(t) = f_{ct}(t'') \wedge f_{ceid}(t) = f_{ceid}(t'')$]
$\therefore$ The *PPCP* = ({*A*},{*B*},{EID_A},{EID_B}) made by *t* cannot merge *t'* and *t''*
$\therefore$ *PPCP* for P_1 is remained in *PSet* (i.e., the set for all *PPCP*s of places) after function *Merge*
$\therefore$ P_1 can be mine by the new algorithm $\Rightarrow$ implicit dependency can be mined.

(2) Mine non-SWF-net (implicit places and *parallel places with multi inputs*):
(i) Mine implicit places

As shown in Fig. 6(c) as well as mentioned in Section 4.3, the pattern of implicit places is the same as implicit dependencies. The proof of mining implicit places is the same as implicit dependencies above.

(ii) Mine parallel places with multi inputs

Fig. 14(a) illustrates the general pattern of *parallel places with multi inputs* (where an *or-join* follows an *or-split* directly and other cases will be discussed in Section 7). Tasks $A_{1...n}$ are the inputs of P_1 and $B_{1...m}$ are the inputs of P_2.

$\because$ (*Parallel places with multi inputs* $\Leftrightarrow$ P_1 and P_2 in Fig. 14(a)) $\wedge$ (Mine Place $\Leftrightarrow$ Make *PPCP*)
$\therefore$ Mine *Parallel places with multi inputs* $\Leftrightarrow$ Construct *PPCP*s for P_1 and P_2
$\because$ $P_1 \in \bullet C \wedge P_2 \in \bullet C$
$\therefore$ Two tokens *t* and *t'* belonging to P_1 and P_2, which are consumed by *C* at the same time satisfy:

$$t = (A_i, EID_{Ai}, C, EID_C) \wedge t' = \left(B_j, EID_{Bj}, C, EID_C\right)$$

$\because$ In different running instances (cases), *C* consumes tokens produced by different pair of tasks.
$\therefore$ All pairs of tokens consumed by C in different cases are in the set *S*:

$$S = \left\{(t_1, t_2)|t_1 = (A_i, EID_{Ai}, C, EID_C) \wedge t_2 = \left(B_j, EID_{Bj}, C, EID_C\right), i \in [1, n], j \in [1, m]\right\}$$

$\because$ $\forall(t_1, t_2) \in S \Rightarrow f_{ct}(t_1) = f_{ct}(t_2) \wedge f_{ceid}(t_1) = f_{ceid}(t_2)$
$\therefore$ $\forall(t_1, t_2) \in S \Rightarrow t_1, t_2$ cannot be merged into the same *PPCP*
$\because$ $\forall i \in [1,n], j \in [1,m] \Rightarrow \exists(t_1 = (A_i, EID_{Ai}, C, EID_C), t_2 = (B_j, EID_{Bj}, C, EID_C)) \in S$
$\therefore$ All tokens produced by A_i and B_j cannot be merged into the same *PPCP* where $i \in [1,n], j \in [1,m]$
$\because$ $\forall t_1, t_2 \in \{(A_i, EID_{Ai}, C, EID_C)| i \in [1,n]\} \Rightarrow f_{pt}(t_1) = f_{pt}(t_2) \wedge f_{peid}(t_1) \neq f_{peid}(t_2)$
$\therefore$ All tokens produced by A_i where $i \in [1,n]$ can be merged into a same *PPCP*. The same to tokens produced by B_j where $j \in [1,m]$.
$\therefore$ Only *PPCP*s for P_1 and P_2 are remained $\Rightarrow$ *Parallel places with multi inputs* can be mined.

(3) Mine one-loop:

$\because$ (One-loop $\Leftrightarrow$ *P* in Fig. 14(b)) $\wedge$ (Mine Place $\Leftrightarrow$ Construct *PPCP* of place)
$\therefore$ Mine one-loop $\Leftrightarrow$ Construct *PPCP* of *P* in Fig. 14(b)
$\because$ *B* or *C* may execute after *A* executing
$\therefore$ Tokens $t_1 = (A, EID_{A1}, B, EID_B)$, $t_2 = (A, EID_{A2}, C, EID_{C1})$ and $t_3 = (B, EID_B, C, EID_{C2})$ are recorded
$\because$ $f_{pt}(t_1) = f_{pt}(t_2) \wedge f_{peid}(t_1) \neq f_{peid}(t_2)$
$\therefore$ t_1 and t_2 form *PPCP* = ({*A*},{*B*,*C*},{EID_{A1},EID_{A2}},{EID_B,EID_{C1}})
$\because$ $f_{ct}(t_3) \in f_{pot}(PPCP) \wedge f_{ceid}(t_3) \notin f_{poeid}(PPCP)$
$\therefore$ t_3 can be merged into *PPCP* and *PPCP* = ({*A*,*B*},{*B*,*C*},{EID_{A1},EID_{A2}, EID_B},{EID_B,EID_{C1},EID_{C2}})
$\therefore$ *P* can be mined $\Rightarrow$ one-loop can be mined.

(4) Mine two-loop:

$\because$ (Two-loop $\Leftrightarrow$ P_1 and P_2 in Fig. 14(c) (*Ne* represents input tasks of P_1 except *B* and *Ne'* represents output tasks of P_2 except *B*)) $\wedge$ (Mine Place $\Leftrightarrow$ Construct *PPCP* of place)

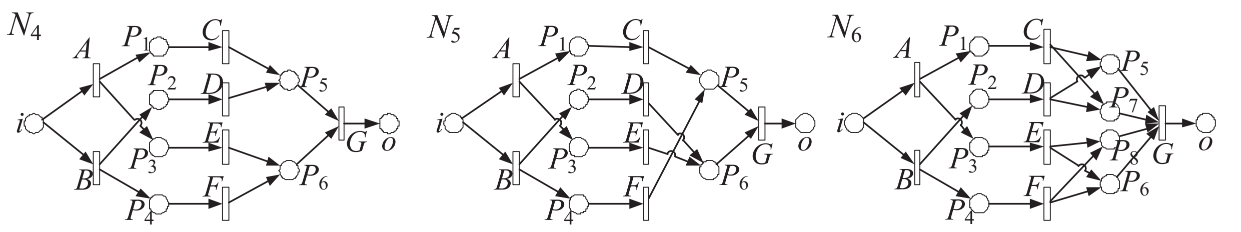

Fig. 15. When mining from the token logs of N_4, τ may output both models N_4 and N_5 but all other algorithms would output N_6.

$\therefore$ Mine two-loop $\Leftrightarrow$ Construct *PPCP* of P_1 and P_2 in Fig. 14(c)
$\because$ *A* followed by *B* firstly and then *B* followed by *A*
$\therefore$ Tokens $t_1 = (Ne, EID_{Ne}, A, EID_{A1})$, $t_2 = (A, EID_{A1}, B, EID_B)$, $t_3 = (B, EID_B, A, EID_{A2})$, and $t_2 = (A, EID_{A2}, Ne', EID_{Ne'})$ are recorded
$\because [f_{ct}(t_1) = f_{ct}(t_3) \wedge f_{ceid}(t_1) \neq f_{ceid}(t_3)] \wedge [f_{pt}(t_2) = f_{ct}(t_4) \wedge f_{peid}(t_2) \neq f_{peid}(t_4)]$
$\therefore t_1$ and t_3 form $PPCP_1 = (\{Ne,B\},\{A\},\{EID_{Ne},EID_B\},\{EID_{A1},EID_{A2}\})$
t_2 and t_4 form $PPCP_2 = (\{A\},\{B,Ne'\},\{EID_{A1},EID_{A2}\},\{EID_B,EID_{Ne'}\})$
$\because PPCP_1$ stands for P_1 and $PPCP_2$ stands for P_2
$\therefore P_1$ and P_2 can be mined $\Rightarrow$ two-loop can be mined. □

Based on the discussion of limitations of the existing process mining algorithms including α, α^+, α^{++}, β and λ, as well as the above proof, we can conclude that the token log based τ algorithm is able to mine all four types of process structures including implicit dependency, non-SWF-net (parallel places with multi inputs and implicit place), one-loop and two-loop, which cannot be fully discovered by any of the existing event log based mining algorithms (α, α^+, α^{++}, β and λ).

7. Discussion of limitations

As mentioned in Section 4.3, there is a limitation of τ in handling the *parallel places with multi inputs* (i.e., the pattern forbidden by SWF-net as shown in Fig. 6(b)). When mining the token log of process model N_4 in Fig. 15, τ may also output the model N_5 as one of its mining results together with the correct one. However, it still outperforms other algorithms (i.e., α, α^+, α^{++}, β and λ) which would simply output the incorrect model as N_6 in Fig. 15.

The model N_4 is considered as a non-Well-structured WF-net and Well-structured has been defined by Aalst et al. in [25] as following.

Definition 15 (Well-handled/well-structured [25]). Petri net $PN = (P,T,F)$ is well-handled *iff* $\forall x,y \in P \cup T$ and one of them is a place, the other is a transition: $\forall C_1,C_2$ are two paths from x to y, if $\alpha(C_1) \cup \alpha(C_2) = \{x, y\}$ then $C_1 = C_2$. And the Workflow net extended from *PN* is well-structured *iff PN* is well-handled.

The method α is for getting the names of all places and transitions in a path. The definition of well-structured says that the *and-split* structure must be followed by an *and-join* and the *or-split* must be followed by an *or-join* in workflow nets. The new mining algorithm τ cannot mine the *parallel places with multi inputs* whose or-joins do not follow an or-split.

In N_4, P_5 and P_6, whose *or-joins* do not follow an *or-split*, become the *parallel places with multi inputs*. N_4 has two kinds of running instances (cases). In one case, tasks *A*, *C*, *E* and *G* are executed. In another case, tasks *B*, *D*, *F* and *G* are executed. The tokens from P_5 and P_6 consumed by *G* in different cases are produced by (*C*, *E*) or (*D*, *F*). There are at least four tokens passing P_5 and P_6 in different cases and let tokens $t_1 = (C, EID_c, G, EID_{g1})$, $t_2 = (E, EID_e, G, EID_{g1})$, $t_3 = (D, EID_d, G, EID_{g2})$, and $t_4 = (F, EID_f, G, EID_{g2})$ be the four tokens consumed by G. Among these four tokens, t_1 and t_2 cannot be merged into the same *PPCP* (pre–post contents for place) because both their consumers and the consumers' *EIDs* are the same and so do t_3 and t_4 (i.e., t_1 can merge with t_3 or t_4 and t_2 can merge with t_3 or t_4). If t_1 merges with t_3 and t_2 merges with t_4, that is, two *PPCPs*, $(\{C,D\},\{G\},\{EID_c, EID_d\},\{EID_{g1}, EID_{g2}\})$ and $(\{E,F\},\{G\},\{EID_e, EID_f\},\{EID_{g1}, EID_{g2}\})$ are created, then P_5 and P_6 in N_4 can be mined. But if t_1 merges with t_4 and t_2 merges with t_3, that is, two *PPCPs*, $(\{C,F\},\{G\},\{EID_c, EID_f\},\{EID_{g1}, EID_{g2}\})$ and $(\{E,D\},\{G\},\{EID_e, EID_d\},\{EID_{g1}, EID_{g2}\})$ are created, then P_5 and P_6 in N_5 can be mined.

Our future work will address the mining of *parallel places with multi inputs* as mentioned in Section 9.

8. Related works

Process mining was initiated by Cook and Wolf [3] and it is sometimes also called Process discovery. One of the objectives of process discovery is to meet the big challenges in modeling the running and complex business processes. To rigorously model the structure and analyze the efficiency of complex, Aalst proposed the WF-net based on Petri nets to model business processes. You can find more properties of WF-nets in [25,32] and more applications of different types of Petri-nets in [10,11,13,14]. Most of the existing process discovery methods were based on event logs recorded during the process executions. These methods can be categorized into two classes based on difference types of event logs: (1) the event logs only contain tasks, such as α [26]; (2) the event logs contain the post tasks of the current executing task, such as λ [28]. In this study, we propose the τ algorithm that initially leverages the data describing the process resource changes which is carried by the token log, for the sake of enhancing the effectiveness and efficiency of process mining.

Process mining has been extended to cover process discovery, conformance checking and process enhancement [22]. Conformance checking is usually performed in terms of the structures and behaviors of processes measured by fitness and appropriateness respectively [18]. As evolved from conformance checking, process mining was also applied on studying the similarity between the structures and behaviors of two process models [5,33]. Besides places and tasks, Passage [20] and Region [1,27] which refer to higher level structures composed by places and tasks in Petri nets are also used in process mining to decompose process models. Besides the event logs which mainly record the execution of tasks in the system, the logs recording timing information [6,34] of the system are also used in process mining. Process mining is also used in other research fields, such as improving software process [12] and supporting personal adaptive learning [16]. The token log and the associated process mining algorithm proposed in this paper aims to record and utilize the data (resources) produced and consumed by the system.

Process mining is a sub field of workflow management. It plays a very important role in synergizing data mining with business process management and brings about a big enhancement on the techniques of Business Intelligence (BI) [22]. Data Mining Technical Committee in the IEEE Computational Intelligence Society has established the IEEE Process Mining Task Force to bring researchers of workflow management together as the emergent needs of leveraging data logs in mining the process models and/or workflows. The workflow reference model [9] defined the features, terms, modules and structures needed in workflow management. A number of data-driven approaches have been used in workflow mining. In [8], Herbst employed a machine learning approach in process mining and discovering workflow models under the context of workflow management. In addition to workflow mining, other aspects of workflow management addressed the organizing, modeling, managing and analyzing the process workflows [23]. Aalst mentioned that there were ten huge problems in the research of workflow management [21,24].

9. Conclusions and future work

In this study, we have proposed a novel perspective to employ the data carried by token log which tracks the changes of process resources for process mining. A new process mining algorithm τ is invented based on the token log to improve the mining efficiency as well as enhancing the mining capability as compared to the traditional process mining methods based on event logs. We have developed three plug-ins on top of the existing workflow engine, process modeling and mining platforms (YAWL, PIPE and ProM) to prove the feasibility of token log and to realize the token log generation and algorithm τ.

We have evaluated the improvements of the mining efficiency of the token-log-based mining algorithm τ in terms of the reduced log size and execution time using the two plug-ins on PIPE and ProM. The results of pairwise *t*-tests show that there is no big difference between the efforts that are taken by the same workflow system to generate the token log and the event log. Nevertheless, the executing efficiencies of event-log-based algorithms α and λ are significantly lower than that of algorithm τ. Besides, no matter how complex a system process model is, the size of its event log used by the existing mining algorithms (α [26], α^+ [4], α^{++} [29], β [30] and λ [28]) is always larger than the size of its token log. The more selections/choices and parallelisms/branches the model contains, the faster its event log size increases than its token log size. Finally, we have demonstrated the ability of algorithm τ in mining the four special structures in Fig. 7 (implicit dependency, non-SWF-net, one-loop and two-loop), which cannot be fully discovered by any of the existing event-log-based mining algorithms including α, α^+, α^{++}, β and λ and a formal proof is provided to generalize the conclusion as well.

Our future work will extend τ algorithm to mine the structure of *parallel places with multi inputs* described in Section 7. We will also test the token-log-based process mining algorithm on various types of real world system processes to make full use of the resource usage data generated in process workflow management.

Acknowledgments

This work was supported by the National 863 Program (2015AA01A203), the National 973 Program (2013CB329102), the National Natural Science Foundation, China (nos.: 61100039, 61272188, 61572162, 61572251), the Natural Science Foundation of Jiangsu Province (no. BK20131277), U.S. National Science Foundation (NSF Awards CNS-1126747), the Open Foundation of State key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (no. SKLNST-2013-1-14), the Open Foundation of State Key Laboratory for Novel Software Technology of Nanjing University (nos.: KFKT2013A21, KFKT2014B15), the Fundamental Research Funds for the Central Universities.

References

[1] J. Carmona, J. Cortadella, New region-based algorithms for deriving bounded petri nets, IEEE Transac. Comput. 59 (3) (2010) 371–384.
[2] K.G. Coman, A.M. Odlyzko, Internet growth: is there a Moore's law for data traffic?, Handbook of Massive Data Sets, Kluwer, 2001, pp. 47–93.
[3] J. Cook, A.L. Wolf, Discovering models of software processes from event-based data, ACM Transac. Softw. Eng. Methodol. (TOSEM) 7 (1998) 215–249.
[4] A. de Mederios, B. van Dongen, W. van der Aalst, T. Weijters, Process Mining: Extending the A; Algorithm to Mine Short Loops, Technical Report, University of Technology, Eindhoven, 2004.
[5] R. Dijkman, M. Dumas, L. Garcia-Banuelos, Graph maching algorithms for business process model similarity search., Business Process Management, LNCS, 5701, 2009, pp. 48–63.
[6] H. Duan, Q. Zeng, H. Wang, S.X. Sun, D. Xu, Classification and evaluation of timed running schemas for workflow based on process mining, J. Syst. Softw. 82 (2009) 400–410.
[7] F. Gorunescu, Data Mining: Concepts, Models and Techniques, Blue Publishing House, ClujNapoca, 2011.

[8] J. Herbst, D. Karagiannis, Integrating machine learning and workflow management to support acquisition and adaptation of workflow models, Ninth International Workshop, Database and Expert Systems Applications, IEEE, 1998, pp. 745–752.
[9] D. Hollingsworth, The workflow reference model, 1995.
[10] Q. Hu, Y. Du, S. Yu, Service net algebra based on logic Petri nets, Inform. Sci. 268 (2014) 271–289.
[11] H. Huang, H. Kirchner, Secure interoperation design in multi-domains environments based on colored Petri nets, Inform. Sci. 221 (2013) 591–606.
[12] H. Huang, J. Xiao, Q. Yang, Q. Wang, H. Wu, Creating process-agents incrementally by mining process asset library, Inform. Sci. 233 (2013) 183–199.
[13] Z. Li, M. Zhou, M.D. Jeng, A maximally permissive deadlock prevention policy for FMS based on Petri net siphon control and the theory of regions, IEEE Transac. Autom. Sci. Eng. 5 (2008) 182–188.
[14] Z. Li, M. Zhou, N. Wu, A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems, IEEE Trans. Syst. Man Cybernet. 38 (2008) 173–188.
[15] T. Murata, Petri nets: properties, analysis and applications, Proc. IEEE 77 (4) (1989) 541–580.
[16] K. Okoye, A.R.H. Tawil, U. Naeem, R. Bashroush, E. Lamine, A semantic rule-based approach supported by process mining for personalised adaptive learning, Proc. Comput. Sci. 37 (2014) 203–210.
[17] W. Reisig, Petri Nets: An Introduction, Springer, Verlag, New York, Inc., New York, NY, USA, 1985.
[18] A. Rozinat, W. van der Aalst, Conformance checking of process based on monitoring real behavior, Inform. Syst. 33 (2008) 64–95.
[19] A. Rozinat, W. van der Aalst, Decision mining in prom, Business Process Management, Lecture Notes in Computer Science, 4102, Springer, Berlin , Heidelberg, 2006, pp. 420–425.
[20] W. van der Aalst, Decomposing process mining problems using passages, Application and Theory of Petri Nets, Lecture Notes in Computer Science, 7347, Springer, 2012, pp. 72–91.
[21] W. van der Aalst, Process discovery: capturing the invisible, Computat. Intell. Mag. 5 (1) (2003) 28–31.
[22] W. van der Aalst, A. Adriansyah, A.K.A. de Medeiros, et al., Process mining manifesto, Business Process Management Workshops, Lecture Notes in Business Information Processing, 100, Springer, 2012, pp. 169–194.
[23] W. van der Aalst, K. van Hee, Workflow Management: Models, Methods, and Systems, MIT Press, Cambridge, MA, 2002.
[24] W. van der Aalst, T. Weijters, Process mining: a research agenda, Comput. Indust. 53 (2004) 231–244.
[25] W. van der Aalst, T. Weijters, L. Maruster, The application of petri nets to workflow management, J. Circuit. Syst. Comput. 8 (1998) 21–66.
[26] W. van der Aalst, T. Weijters, L. Maruster, Workflow mining: discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (2004) 1128–1142.
[27] J.M. van der Werf, B. van Dongen, K. van Hee, C. Hurkens, A. Serebrenik, Process discover using integer linear programming, Applications and Theory of Petri Nets, Lecture Notes in Computer Science, 5062, Springer, 2009, pp. 368–387.
[28] D. Wang, J. Ge, H. Hu, B. Luo, L. Huang, Discovering process models from event multiset, Exp. Syst. Appl. 39 (2012) 11970–11978.
[29] L. Wen, W. van der Aalst, J. Wang, J. Sun, Mining process models with non-free-choice constructs, Data Min. Knowl. Discov. 15 (2007) 145–180.
[30] L. Wen, J. Wang, W. van der Aalst, B. Huang, J. Sun, A novel approach for process mining based on event types, J. Intellig. Inform. Syst. 32 (2009) 163–190.
[31] L. Wen, J. Wang, W. van der Aalst, B. Huang, J. Sun, Mining process models with prime invisible tasks, Data Knowl. Eng. 69 (2010) 999–1021.
[32] M.T. Wynn, H.M.W. Verbeek, W.M.P. van der Aalst, A.H.M. ter Hofstede, D. Edmond, Soundness-preserving reduction rules for reset workflow nets, Inform. Sci. 179 (2009) 769–790.
[33] Z. Yan, R. Dijkman, P. Grefen, Fast business process similarity search with feature-based similarity estima-tion, On the Move to Meaningful Internet Systems: OTM2010, Lecture Notes in Computer Science, 6426, Springer, 2010, pp. 60–77.
[34] Q. Zeng, H. Wang, D. Xu, H. Duan, Y. Han, Conflict detection and resolution for workflows constrained by resources and non-determined durations, J. Syst. Softw. 81 (2008) 1491–1504.