Workflow Mining: Discovering Process Models from Event Logs

Wil van der Aalst, Ton Weijters, and Laura Maruster

Abstract—Contemporary workflow management systems are driven by explicit process models, i.e., a completely specified workflow design is required in order to enact a given workflow process. Creating a workflow design is a complicated time-consuming process and, typically, there are discrepancies between the actual workflow processes and the processes as perceived by the management. Therefore, we have developed techniques for discovering workflow models. The starting point for such techniques is a so-called "workflow log" containing information about the workflow process as it is actually being executed. We present a new algorithm to extract a process model from such a log and represent it in terms of a Petri net. However, we will also demonstrate that it is not possible to discover arbitrary workflow processes. In this paper, we explore a class of workflow processes that can be discovered. We show that the α-algorithm can successfully mine any workflow represented by a so-called SWF-net.

Index Terms—Workflow mining, workflow management, data mining, Petri nets.

✦

---

1 INTRODUCTION

DURING the last decade, workflow management concepts and technology [3], [5], [15], [26], [28] have been applied in many enterprise information systems. Workflow management systems such as Staffware, IBM MQSeries, COSA, etc., offer generic modeling and enactment capabilities for structured business processes. By making graphical process definitions, i.e., models describing the life-cycle of a typical case (workflow instance) in isolation, one can configure these systems to support business processes. Besides pure workflow management systems, many other software systems have adopted workflow technology. Consider, for example, ERP (Enterprise Resource Planning) systems such as SAP, PeopleSoft, Baan and Oracle, CRM (Customer Relationship Management) software, etc. Despite its promise, many problems are encountered when applying workflow technology. One of the problems is that these systems require a workflow design, i.e., a designer has to construct a detailed model accurately describing the routing of work. Modeling a workflow is far from trivial: It requires deep knowledge of the workflow language and lengthy discussions with the workers and management involved.

Instead of starting with a workflow design, we start by gathering information about the workflow processes as they take place. We assume that it is possible to record events such that

1. each event refers to a task (i.e., a well-defined step in the workflow),
2. each event refers to a case (i.e., a workflow instance), and

---

● *The authors are with the Department of Technology Management, Eindhoven University of Technology, PO Box 513, NL-5600 MB, Eindhoven, The Netherlands.*
E-mail: {w.m.p.v.d.aalst, A.J.M.M.Weijters, l.maruster}@tm.tue.nl.

3. events are totally ordered (i.e., in the log events are recorded sequentially, even though tasks may be executed in parallel).

Any information system using transactional systems such as ERP, CRM, or workflow management systems will offer this information in some form. Note that we do not assume the presence of a workflow management system. The only assumption we make is that it is possible to collect workflow logs with event data. These workflow logs are used to construct a process specification which adequately models the behavior registered. We use the term *process mining* for the method of distilling a structured process description from a set of real executions.

To illustrate the principle of process mining, we consider the workflow log shown in Table 1. This log contains information about five cases (i.e., workflow instances). The log shows that for four cases (1, 2, 3, and 4), the tasks A, B, C, and D have been executed. For the fifth case, only three tasks are executed: tasks A, E, and D. Each case starts with the execution of A and ends with the execution of D. If task B is executed, then task C is also executed. However, for some cases, task C is executed before task B. Based on the information shown in Table 1 and by making some assumptions about the completeness of the log (i.e., assuming that the cases are representative and a sufficient large subset of possible behaviors is observed), we can deduce for example the process model shown in Fig. 1. The model is represented in terms of a Petri net [39]. The Petri net starts with task A and finishes with task D. These tasks are represented by transitions. After executing A, there is a choice between either executing B and C in parallel, or just executing task E. To execute B and C in parallel, two nonobservable tasks (AND-split and AND-join) have been added. These tasks have been added for routing purposes only and are not present in the workflow log. Note that we assume that two tasks are in parallel if they appear in any order. However, by distinguishing between start events and end events for tasks, it is possible to explicitly detect

TABLE 1
A Workflow Log

case identifier	task identifier
case 1	task A
case 2	task A
case 3	task A
case 3	task B
case 1	task B
case 1	task C
case 2	task C
case 4	task A
case 2	task B
case 2	task D
case 5	task A
case 4	task C
case 1	task D
case 3	task C
case 3	task D
case 4	task B
case 5	task E
case 5	task D
case 4	task D

parallelism. Start events and end events can also be used to indicate that tasks take time. However, to simplify the presentation, we assume tasks to be atomic without losing generality. In fact, in our tool EMiT [4], we refine this even further and assume a customizable transaction model for tasks involving events like "start task," "withdraw task," "resume task," "complete task," etc. [4]. Nevertheless, it is important to realize that such an approach only works if events like these are recorded at the time of their occurrence.

The basic idea behind process mining, also referred to as workflow mining, is to construct Fig. 1 from the information given in Table 1. In this paper, we will present a new algorithm and prove its correctness.

Process mining is useful for at least two reasons. First of all, it could be used as a tool to find out how people and/or procedures really work. Consider, for example, processes supported by an ERP system like SAP (e.g., a procurement process). Such a system logs all transactions, but in many cases does not enforce a specific way of working. In such an environment, process mining could be used to gain insight in the actual process. Another example would be the flow of patients in a hospital. Note that in such an environment, all activities are logged, but information about the underlying process is typically missing. In this context, it is important

to stress that management information systems provide information about key performance indicators like resource utilization, flow times, and service levels, but *not* about the underlying business processes (e.g., causal relations, ordering of activities, etc.). Second, process mining could be used for *Delta analysis*, i.e., comparing the actual process with some predefined process. Note that in many situations, there is a descriptive or prescriptive process model. Such a model specifies how people and organizations are assumed/expected to work. By comparing the descriptive or prescriptive process model with the discovered model, discrepancies between both can be detected and used to improve the process. Consider, for example, the so-called reference models in the context of SAP. These models describe how the system should be used. Using process mining, it is possible to verify whether this is the case. In fact, process mining could also be used to compare different departments/organizations using the same ERP system.

An additional benefit of process mining is that information about the way people and/or procedures really work and differences between actual processes and predefined processes can be used to trigger Business Process Reengineering (BPR) efforts or to configure "process-aware information systems" (e.g., workflow, ERP, and CRM systems).

Table 1 contains the minimal information we assume to be present. In many applications, the workflow log contains a timestamp for each event and this information can be used to extract additional causality information. Moreover, we are also interested in the relation between attributes of the case and the actual route taken by a particular case. For example, when handling traffic violations: Is the make of a car relevant for the routing of the corresponding traffic violations? (For example, "People driving a Ferrari always pay their fines in time.")

For this simple example, it is quite easy to construct a process model that is able to regenerate the workflow log. For larger workflow models this is much more difficult. For example, if the model exhibits alternative and parallel routing, then the workflow log will typically not contain all possible combinations. Consider 10 tasks which can be executed in parallel. The total number of interleavings is 10! = 3628800. It is not realistic that each interleaving is present in the log. Moreover, certain paths through the process model may have a low probability and, therefore, remain undetected. Noisy data (i.e., logs containing rare events, exceptions, and/or incorrectly recorded data) can further complicate matters.

In this paper, we do not focus on issues such as noise. We assume that there is no noise and that the workflow log
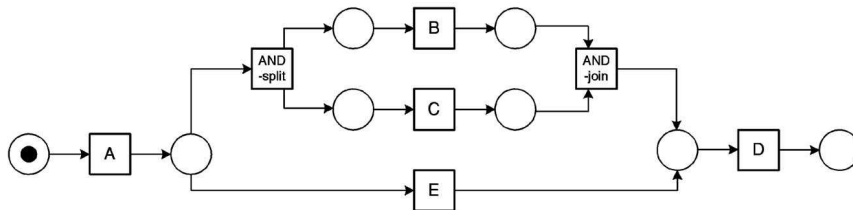

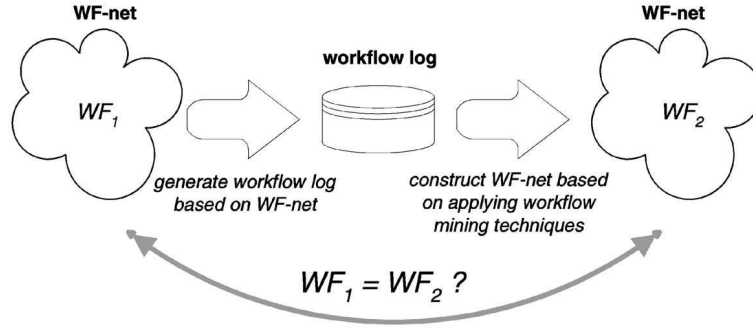Fig. 1. A process model corresponding to the workflow log.

Fig. 2. The rediscovery problem: For which class of WF-nets is it guaranteed that WF_2 is equivalent to WF_1?

contains "sufficient" information. Under these ideal circumstances, we investigate whether it is possible to *rediscover* the workflow process, i.e., for which class of workflow models is it possible to accurately construct the model by merely looking at their logs. This is not as simple as it seems. Consider, for example, the process model shown in Fig. 1. The corresponding workflow log shown in Table 1 does not show any information about the AND-split and the AND-join. Nevertheless, they are needed to accurately describe the process. These and other problems are addressed in this paper. For this purpose, we use *workflow nets* (WF-nets). WF-nets are a class of Petri nets specifically tailored toward workflow processes. Fig. 1 shows an example of a WF-net.

To illustrate the *rediscovery problem* we use Fig. 2. Suppose we have a log based on many executions of the process described by a WF-net WF_1. Based on this workflow log and using a mining algorithm, we construct a WF-net WF_2. An interesting question is whether $WF_1 = WF_2$. In this paper, we explore the class of WF-nets for which $WF_1 = WF_2$. Note that the rediscovery problem is only addressed to explore the theoretical limits of process mining and to test the algorithm presented in this paper. We have used these results to develop tools that can discover unknown processes and have successfully applied these tools to mine real processes.

The remainder of this paper is organized as follows: First, we introduce some preliminaries, i.e., Petri nets and WF-nets. In Section 3, we formalize the problem addressed in this paper. Section 4 discusses the relation between causality detected in the log and places connecting transitions in the WF-net. Based on these results, an algorithm for process mining is presented. The quality of this algorithm is supported by the fact that it is able to rediscover a large class of workflow processes. The paper finishes with an overview of related work and some conclusions.

2 PRELIMINARIES

This section introduces the techniques used in the remainder of this paper. First, we introduce standard Petri-net notations, then we define the class of WF-nets.

2.1 Petri Nets

We use a variant of the classic Petri-net model, namely, Place/Transition nets. For an elaborate introduction to Petri nets, the reader is referred to [12], [37], [39].

Definition 2.1 (P/T-nets)[1]. *An Place/Transition net, or simply P/T-net, is a tuple* (P, T, F), *where:*

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

A marked *P/T-net is a pair* (N, s), *where* $N = (P, T, F)$ *is a P/T-net and where* s *is a bag over* P *denoting the* marking *of the net. The set of all marked P/T-nets is denoted* $\mathcal{N}$.

A marking is a *bag* over the set of places P, i.e., it is a function from P to the natural numbers. We use square brackets for the enumeration of a bag, e.g., $[a^2, b, c^3]$ denotes the bag with two as, one b, and three cs. The sum of two bags $(X + Y)$, the difference $(X - Y)$, the presence of an element in a bag $(a \in X)$, and the notion of subbags $(X \leq Y)$ are defined in a straightforward way and they can handle a mixture of sets and bags.

Let $N = (P, T, F)$ be a P/T-net. Elements of $P \cup T$ are called *nodes*. A node x is an *input node* of another node y iff there is a directed arc from x to y (i.e., $(x, y) \in F$). Node x is an *output node* of y iff $(y, x) \in F$. For any $x \in P \cup T$, $\overset{N}{\bullet}x = \{y \mid (y, x) \in F\}$ and $x\overset{N}{\bullet} = \{y \mid (x, y) \in F\}$; the superscript N may be omitted if clear from the context.

Fig. 1 shows a P/T-net consisting of eight places and seven transitions. Transition A has one input place and one output place, transition *AND-split* has one input place and two output places, and transition *AND-join* has two input places and one output place. The black dot in the input place of A represents a token. This token denotes the initial marking. The dynamic behavior of such a marked P/T-net is defined by a *firing rule*.

Definition 2.2 (Firing rule). *Let* $(N = (P, T, F), s)$ *be a marked P/T-net. Transition* $t \in T$ *is* enabled, *denoted* $(N, s)[t\rangle$, *iff* $\bullet t \leq s$. *The* firing rule $\_[\_\rangle\_ \subseteq \mathcal{N} \times T \times \mathcal{N}$ *is the smallest relation satisfying for any* $(N = (P, T, F), s) \in \mathcal{N}$ *and any* $t \in T$, $(N, s)[t\rangle \Rightarrow (N, s)[t\rangle(N, s - \bullet t + t\bullet)$.

In the marking shown in Fig. 1 (i.e., one token in the source place), transition A is enabled and firing this transition removes the token from the input place and puts a token in the output place. In the resulting marking, two

---

transitions are enabled: *E* and *AND-split*. Although both are enabled, only one can fire. If *AND-split* fires, one token is consumed and two tokens are produced.

Definition 2.3 (Reachable markings). *Let* (N, s_0) *be a marked P/T-net in* $\mathcal{N}$*. A marking* s *is* reachable *from the initial marking* s_0 *iff there exists a sequence of enabled transitions whose firing leads from* s_0 *to* s*. The set of reachable markings of* (N, s_0) *is denoted* $[N, s_0\rangle$*.*

The marked P/T-net shown in Fig. 1 has eight reachable markings. Sometimes, it is convenient to know the sequence of transitions that are fired in order to reach some given marking. This paper uses the following notations for sequences. Let A be some alphabet of identifiers. A *sequence of length* n, for some natural number $n \in \mathbb{N}$, over alphabet A is a function $\sigma : \{0, \ldots, n-1\} \to A$. The sequence of length zero is called the empty sequence and written ε. For the sake of readability, a sequence of positive length is usually written by juxtaposing the function values: For example, a sequence $\sigma = \{(0, a), (1, a), (2, b)\}$, for $a, b \in A$, is written aab. The set of all sequences of arbitrary length over alphabet A is written A^*.

Definition 2.4 (Firing sequence). *Let* (N, s_0) *with* $N = (P, T, F)$ *be a marked P/T net. A sequence* $\sigma \in T^*$ *is called a* firing sequence *of* (N, s_0) *iff, for some natural number* $n \in \mathbb{N}$*, there exist markings* $s_1, \ldots, s_n$ *and transitions* $t_1, \ldots, t_n \in T$ *such that* $\sigma = t_1 \ldots t_n$ *and, for all* i *with* $0 \leq i < n$*,* $(N, s_i)[t_{i+1}\rangle$ *and* $s_{i+1} = s_i - \bullet t_{i+1} + t_{i+1} \bullet$*. (Note that* $n = 0$ *implies that* $\sigma = \varepsilon$ *and that* ε *is a firing sequence of* (N, s_0)*.) Sequence* σ *is said to be* enabled *in marking* s_0*, denoted* $(N, s_0)[\sigma\rangle$*. Firing the sequence* σ *results in a marking* s_n*, denoted* $(N, s_0)[\sigma\rangle(N, s_n)$*.*

Definition 2.5 (Connectedness). *A net* $N = (P, T, F)$ *is* weakly connected, *or simply* connected, *iff, for every two nodes* x *and* y *in* $P \cup T$*,* $x(F \cup F^{-1})^* y$*, where* R^{-1} *is the inverse and* R^* *the reflexive and transitive closure of a relation* R*. Net* N *is* strongly connected *iff, for every two nodes* x *and* y*,* $xF^* y$*.*

We assume that all nets are weakly connected and have at least two nodes. The P/T-net shown in Fig. 1 is connected, but not strongly connected because there is no directed path from the sink place to the source place, or from D to A, etc.

Definition 2.6 (Boundedness, safeness). *A marked net* $(N = (P, T, F), s)$ *is* bounded *iff the set of reachable markings* $[N, s\rangle$ *is finite. It is* safe *iff, for any* $s' \in [N, s\rangle$ *and any* $p \in P$*,* $s'(p) \leq 1$*. Note that safeness implies boundedness.*

The marked P/T-net shown in Fig. 1 is safe (and, therefore, also bounded) because none of the eight reachable states puts more than one token in a place.

Definition 2.7 (Dead transitions, liveness). *Let* $(N = (P, T, F), s)$ *be a marked P/T-net. A transition* $t \in T$ *is* dead *in* (N, s) *iff there is no reachable marking* $s' \in [N, s\rangle$ *such that* $(N, s')[t\rangle$*.* (N, s) *is* live *iff, for every reachable marking* $s' \in [N, s\rangle$ *and* $t \in T$*, there is a reachable marking* $s'' \in [N, s'\rangle$ *such that* $(N, s'')[t\rangle$*. Note that liveness implies the absence of dead transitions.*

None of the transitions in the marked P/T-net shown in Fig. 1 is dead. However, the marked P/T-net is not live since it is not possible to enable each transition continuously.

2.2 Workflow Nets

Most workflow systems offer standard building blocks such as the AND-split, AND-join, OR-split, and OR-join [5], [15], [26], [28]. These are used to model sequential, conditional, parallel, and iterative routing (WFMC [15]). Clearly, a Petri net can be used to specify the routing of cases. *Tasks* are modeled by transitions and causal dependencies are modeled by places and arcs. In fact, a place corresponds to a *condition* which can be used as pre and/or postcondition for tasks. An AND-split corresponds to a transition with two or more output places, and an AND-join corresponds to a transition with two or more input places. OR-splits/OR-joins correspond to places with multiple outgoing/ingoing arcs. Given the close relation between tasks and transitions, we use the terms interchangeably.

A Petri net which models the control-flow dimension of a workflow, is called a *WorkFlow net*(WF-net). It should be noted that a WF-net specifies the dynamic behavior of a single case in isolation.

Definition 2.8 (Workflow nets). *Let* $N = (P, T, F)$ *be a P/T-net and* $\bar{t}$ *a fresh identifier not in* $P \cup T$*.* N *is a* workflow net *(WF-net) iff:*

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

The P/T-net shown in Fig. 1 is a WF-net. Note that, although the net is not strongly connected, the *short-circuited* net $\overline{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ (i.e., the net with transition $\bar{t}$ connecting o to i) is strongly connected. Even if a net meets all the syntactical requirements stated in Definition 2.8, the corresponding process may exhibit errors such as deadlocks, tasks which can never become active, livelocks, garbage being left in the process after termination, etc. Therefore, we define the following correctness criterion.

Definition 2.9 (Sound). *Let* $N = (P, T, F)$ *be a WF-net with input place* i *and output place* o*.* N *is* sound *iff:*

1. *safeness:* $(N, [i])$ *is safe,*
2. *proper completion: for any marking* $s \in [N, [i]\rangle$*,* $o \in s$ *implies* $s = [o]$*,*
3. *option to complete: for any marking* $s \in [N, [i]\rangle$*,* $[o] \in [N, s\rangle$*, and*
4. *absence of dead tasks:* $(N, [i])$ *contains no dead transitions.*

The set of all sound WF-nets is denoted $\mathcal{W}$*.*

The WF-net shown in Fig. 1 is sound. Soundness can be verified using standard Petri-net-based analysis techniques. In fact, soundness corresponds to liveness and safeness of the corresponding short-circuited net [1], [2], [5]. This way, efficient algorithms and tools can be applied. An example of a tool tailored toward the analysis of WF-nets is Woflan [47].

3   THE REDISCOVERY PROBLEM

After introducing some preliminaries, we return to the topic of this paper: *workflow mining*. The goal of workflow mining is to find a workflow model (e.g., a WF-net) on the basis of a *workflow log*. Table 1 shows an example of a workflow log. Note that the ordering of events within a case is relevant, while the ordering of events among cases is of no importance. Therefore, we define a workflow log as follows.

Definition 3.1 (Workflow trace, Workflow log). *Let T be a set of tasks. $\sigma \in T^*$ is a* workflow trace *and $W \in \mathcal{P}(T^*)$ is a* workflow log.[2]

The workflow trace of case 1 in Table 1 is $ABCD$. The workflow log corresponding to Table 1 is

$$\{ABCD, ACBD, AED\}.$$

Note that in this paper, we abstract from the identity of cases. Clearly, the identity and the attributes of a case are relevant for workflow mining. However, for the theoretical results in this paper, we can abstract from this. For similar reasons, we abstract from the frequency of workflow traces. In Table 1, workflow trace $ABCD$ appears twice (case 1 and case 3), workflow trace $ACBD$ also appears twice (case 2 and case 4), and workflow trace AED (case 5) appears only once. These frequencies are not registered in the workflow log $\{ABCD, ACBD, AED\}$. Note that when dealing with noise, frequencies are of the utmost importance. However, in this paper, we do not deal with issues such as noise. Therefore, this abstraction is made to simplify notation. For readers interested in how we deal with noise and related issues, we refer to [31], [32], [48], [49], [50].

To find a workflow model on the basis of a workflow log, the log should be analyzed for causal dependencies, e.g., if a task is always followed by another task, it is likely that there is a causal relation between both tasks. To analyze these relations, we introduce the following notations.

Definition 3.2 (Log-based ordering relations). *Let W be a workflow log over T, i.e., $W \in \mathcal{P}(T^*)$. Let $a, b \in T$:*

- *$a >_W b$ iff there is a trace $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ and $i \in \{1, \ldots, n-2\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$,*
- *$a \rightarrow_W b$ iff $a >_W b$ and $b \not>_W a$,*
- *$a \#_W b$ iff $a \not>_W b$ and $b \not>_W a$, and*
- *$a \|_W b$ iff $a >_W b$ and $b >_W a$.*

Consider the workflow log $W = \{ABCD, ACBD, AED\}$ (i.e., the log shown in Table 1). Relation $>_W$ describes which tasks appeared in sequence (one directly following the other). Clearly, $A >_W B$, $A >_W C$, $A >_W E$, $B >_W C$, $B >_W D$, $C >_W B$, $C >_W D$, and $E >_W D$. Relation $\rightarrow_W$ can be computed from $>_W$ and is referred to as the *(direct) causal relation* derived from workflow log W. $A \rightarrow_W B$, $A \rightarrow_W C$, $A \rightarrow_W E$, $B \rightarrow_W D$, $C \rightarrow_W D$, and $E \rightarrow_W D$. Note that $B \not\rightarrow_W C$ because $C >_W B$. Relation $\|_W$ suggests potential parallelism. For log W, tasks B and C seem to be in parallel, i.e., $B \|_W C$ and $C \|_W B$. If two tasks can follow each other directly in any order, then all possible interleavings are present and,

therefore, they are likely to be in parallel. Relation $\#_W$ gives pairs of transitions that never follow each other directly. This means that there are no direct causal relations and parallelism is unlikely.

Property 3.1. *Let W be a workflow log over T. For any $a, b \in T$: $a \rightarrow_W b$, or $b \rightarrow_W a$, or $a \#_W b$, or $a \|_W b$. Moreover, the relations $\rightarrow_W$, $\rightarrow_W^{-1}$, $\#_W$, and $\|_W$ are mutually exclusive and partition $T \times T$.[3]*

This property can easy be verified. Note that

$$\rightarrow_W = (>_W \setminus >_W^{-1}), \rightarrow_W^{-1} = (>_W^{-1} \setminus >_W),$$

$\#_W = (T \times T) \setminus (>_W \cup >_W^{-1})$, $\|_W = (>_W \cap >_W^{-1})$. Therefore, $T \times T = \rightarrow_W \cup \rightarrow_W^{-1} \cup \#_W \cup \|_W$. If no confusion is possible, the subscript W is omitted.

To simplify the use of logs and sequences, we introduce some additional notations.

Definition 3.3 ($\in$, *first*, *last*). *Let A be a set, $a \in A$, and $\sigma = a_1 a_2 \ldots a_n \in A^*$ a sequence over A of length n. $\in$, first, and last are defined as follows:*

1. *$a \in \sigma$ iff $a \in \{a_1, a_2, \ldots a_n\}$,*
2. *$first(\sigma) = a_1$, if $n \geq 1$, and*
3. *$last(\sigma) = a_n$, if $n \geq 1$.*

To reason about the quality of a workflow mining algorithm, we need to make assumptions about the completeness of a log. For a complex process, a handful of traces will not suffice to discover the exact behavior of the process. Relations $\rightarrow_W$, $\rightarrow_W^{-1}$, $\#_W$, and $\|_W$ will be crucial information for any workflow-mining algorithm. Since these relations can be derived from $>_W$, we assume the log to be complete with respect to this relation.

Definition 3.4 (Complete workflow log). *Let $N = (P, T, F)$ be a sound WF-net, i.e., $N \in \mathcal{W}$. W is a* workflow log of N *iff $W \in \mathcal{P}(T^*)$ and every trace $\sigma \in W$ is a firing sequence of N starting in state $[i]$ and ending in $[o]$, i.e., $(N, [i])[\sigma\rangle(N, [o])$. W is a* complete *workflow log of N iff 1) for any workflow log W' of N: $>_{W'} \subseteq >_W$, and 2) for any $t \in T$ there is a $\sigma \in W$ such that $t \in \sigma$.*

A workflow log of a sound WF-net only contains behaviors that can be exhibited by the corresponding process. A workflow log is complete if all tasks that potentially directly follow each other, in fact, directly follow each other in some trace in the log. Note that transitions that connect the input place i of a WF-net to its output place o are "invisible" for $>_W$. Therefore, the second requirement has been added. If there are no such transitions, this requirement can be dropped as is illustrated by the following property.

Property 3.2. *Let $N = (P, T, F)$ be a sound WF-net. If W is a complete workflow log of N, then*

$$\{t \in T \mid \exists_{t' \in T} t >_W t' \vee t' >_W t\} = \{t \in T \mid t \notin i \bullet \cap \bullet o\}.$$

Proof. Consider a transition $t \in T$. Since N is sound there is firing sequence containing t. If $t \in i \bullet \cap \bullet o$, then this

---

2. $\mathcal{P}(T^*)$ is the powerset of T^*, i.e., $W \subseteq T^*$.

3. $\rightarrow_W^{-1}$ is the inverse of relation $\rightarrow_W$, i.e., $\rightarrow_W^{-1} = \{(y, x) \in T \times T \mid x \rightarrow_W y\}$.

sequence has length 1 and t cannot appear in $>_W$ because this is the only firing sequence containing t. If $t \notin i \bullet \cap \bullet o$, then the sequence has at least length 2, i.e., t is directly preceded or followed by a transition and, therefore, appears in $>_W$.   □

The definition of completeness given in Definition 3.4 may seem arbitrary, but it is not. Note that it would be unrealistic to assume that all possible firing sequences are present in the log. First of all, the number of possible sequences may be infinite (in case of loops). Second, parallel processes typically have an exponential number of states and, therefore, the number of possible firing sequences may be enormous. Finally, even if there is no parallelism and no loops but just N binary choices, the number of possible sequences may be 2^N. Therefore, we need a weaker notion of completeness. If there is no parallelism and no loops but just N binary choices, the number of cases required may be as little as 2 using our notion of completeness. Of course, for a large N, it is unlikely that all choices are observed in just two cases, but it still indicates that this requirement is considerably less demanding than observing all possible sequences. The same holds for processes with loops and parallelism. If a process has N sequential fragments which each exhibit parallelism, the number of cases needed to observe all possible combinations is exponential in the number of fragments. Using our notion of completeness, this is not the case. One could consider even weaker notions of completeness, however, as will be shown in the remainder, even this notion of completeness (i.e., Definition 3.4) is in some situations too weak to detect certain advanced routing patterns.

We will formulate the rediscovery problem introduced in Section 1 assuming a complete workflow log as described in Definition 3.4. Before formulating this problem, we define what it means for a WF-net to be rediscovered.

Definition 3.5 (Ability to rediscover). *Let $N = (P, T, F)$ be a sound WF-net, i.e., $N \in \mathcal{W}$, and let α be a mining algorithm which maps workflow logs of N onto sound WF-nets, i.e., $\alpha : \mathcal{P}(T^*) \to \mathcal{W}$. If for any complete workflow log W of N, the mining algorithm returns N (modulo renaming of places), then α is able to* rediscover *N.*

Note that no mining algorithm is able to find names of places. Therefore, we ignore place names, i.e., α is able to rediscover N iff $\alpha(W) = N$ modulo renaming of places.

The goal of this paper is twofold. First of all, we are looking for a mining algorithm that is able to rediscover sound WF-nets, i.e., based on a complete workflow log, the corresponding workflow process model can be derived. Second, given such an algorithm, we want to indicate the class of workflow nets which can be rediscovered. Clearly, this class should be as large as possible. Note that there is no mining algorithm which is able to rediscover all sound WF-nets. For example, if in Fig. 1 we add a place p connecting transitions A and D, there is no mining algorithm able to detect p since this place is implicit, i.e., the addition of the place does not change the behavior of the net and, therefore, is not visible in the log.

To conclude, we summarize the *rediscovery problem*: "Find a mining algorithm able to rediscover a large class of sound WF-nets on the basis of complete workflow logs." This problem was illustrated in the introduction using Fig. 2.

4 WORKFLOW MINING

In this section, the rediscovery problem is tackled. Before we present a mining algorithm able to rediscover a large class of sound WF-nets, we investigate the relation between the causal relations detected in the log (i.e., $\to_W$) and the presence of places connecting transitions. First, we show that causal relations in $\to_W$ imply the presence of places. Then, we explore the class of nets for which the reverse also holds. Based on these observations, we present a mining algorithm.

4.1 Causal Relations Imply Connecting Places

If there is a causal relation between two transitions according to the workflow log, then there has to be a place connecting these two transitions.

Theorem 4.1. *Let $N = (P, T, F)$ be a sound WF-net and let W be a complete workflow log of N. For any $a, b \in T$: $a \to_W b$ implies $a \bullet \cap \bullet b \neq \emptyset$.*

Proof. Assume $a \to_W b$ and $a \bullet \cap \bullet b = \emptyset$. We will show that this leads to a contradiction and, thus, prove the theorem. Since $a > b$, there is a firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ and $i \in \{1, \ldots, n-2\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$. Let s be the state just before firing a, i.e., $(N, [i])[\sigma'\rangle(N, s)$ with $\sigma' = t_1 \ldots t_{i-1}$. Let s' be the marking after firing b in state s, i.e., $(N, s)[b\rangle(N, s')$. Note that b is enabled in s because it is enabled after firing a and $a \bullet \cap \bullet b = \emptyset$ (i.e., a does not produce tokens for any of the input places of b). a cannot be enabled in s'; otherwise, $b > a$ and not $a \to_W b$. Since a is enabled in s but not in s', b consumes a token from an input place of a and does not return it, i.e., $((\bullet b) \setminus (b\bullet)) \cap \bullet a \neq \emptyset$. There is a place p such that $p \in \bullet a$, $p \in \bullet b$, and $p \notin b \bullet$. Moreover, $a \bullet \cap \bullet b = \emptyset$. Therefore, $p \notin a \bullet$. Since the net is safe, p contains precisely one token in marking s. This token is consumed by $t_i = a$ and not returned. Hence, b cannot be enabled after firing t_i. Therefore, σ cannot be a firing sequence of N starting in i.   □

Let

$$N_1 = (\{i, p_1, p_2, p_3, p_4, o\}, \{A, B, C, D\}, \{(i, A), (A, p_1), (A, p_2),$$
$$(p_1, B), (B, p_3), (p_2, C), (C, p_4), (p_3, D), (p_4, D), (D, o)\}).$$

(This is the WF-net with B and C in parallel, see N_1 in Fig. 4) $W_1 = \{ABCD, ACBD\}$ is a complete log over N_1. Since $A \to_{W_1} B$, there has to be a place between A and B. This place corresponds to p_1 in N_1. Let

$$N_2 = (\{i, p_1, p_2, o\}, \{A, B, C, D\}, \{(i, A), (A, p_1), (p_1, B),$$
$$(B, p_2), (p_1, C), (C, p_2), (p_2, D), (D, o)\}).$$

(This is the WF-net with a choice between B and C, see N_2 in Fig. 4.) $W_2 = \{ABD, ACD\}$ is a complete log over N_2. Since $A \to_{W_2} B$, there has to be a place between A and B. Similarly, $A \to_{W_2} C$ and, therefore, there has to be a place
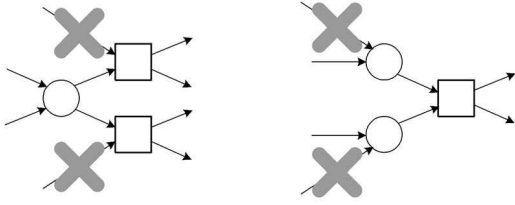
Fig. 3. Two constructs not allowed in SWF-nets.

between A and C. Both places correspond to p_1 in N_1. Note that in the first example (N_1/W_1), the two causal relations $A \to_{W_1} B$ and $A \to_{W_1} C$ correspond to two different places, while in the second example, the two causal relations $A \to_{W_1} B$ and $A \to_{W_1} C$ correspond to a single place.

4.2 Connecting Places "Often" Imply Causal Relations

In this section, we investigate which places can be detected by simply inspecting the log. Clearly, not all places can be detected. For example, places may be implicit which means that they do not affect the behavior of the process. These places remain undetected. Therefore, we limit our investigation to WF-nets without implicit places.

Definition 4.1 (Implicit place). *Let $N = (P, T, F)$ be a P/T-net with initial marking s. A place $p \in P$ is called implicit in (N, s) iff, for all reachable markings $s' \in [N, s\rangle$ and transitions $t \in p \bullet$, $s' \geq \bullet t \setminus \{p\} \Rightarrow s' \geq \bullet t$.*

Fig. 1 contains no implicit places. However, as indicated before, adding a place p connecting transition A and D yields an implicit place. No mining algorithm is able to detect p since the addition of the place does not change the behavior of the net and, therefore, is not visible in the log.

For the rediscovery problem, it is very important that the structure of the WF-net clearly reflects its behavior. Therefore, we also rule out the constructs shown in Fig. 3. The left construct illustrates the constraint that choice and synchronization should never meet. If two transitions share an input place and, therefore, "fight" for the same token, they should not require synchronization. This means that choices (places with multiple output transitions) should not be mixed with synchronizations. The right-hand construct in Fig. 3 illustrates the constraint that if there is a synchronization, all preceding transitions should have fired, i.e., it is not allowed to have synchronizations directly preceded by an OR-join. WF-nets which satisfy these requirements are named *structured workflow nets*.

Definition 4.2 (SWF-net). *A WF-net $N = (P, T, F)$ is an SWF-net (Structured workflow net) iff:*

1. *For all $p \in P$ and $t \in T$ with $(p, t) \in F$: $|p \bullet| > 1$ implies $| \bullet t| = 1$.*
2. *For all $p \in P$ and $t \in T$ with $(p, t) \in F$: $| \bullet t| > 1$ implies $| \bullet p| = 1$.*
3. *There are no implicit places.*

At first sight, the three requirements in Definition 4.2 seem quite restrictive. From a practical point of view, this is not the case. First of all, SWF-nets allow for all routing constructs encountered in practice, i.e., sequential, parallel,
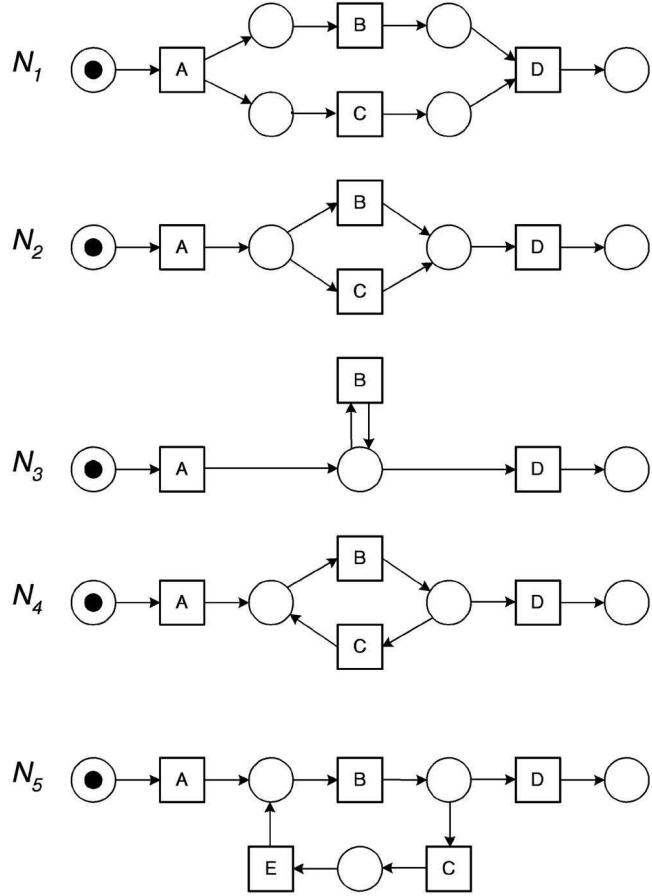

Fig. 4. Five sound SWF-nets.

conditional, and iterative routing are possible and the basic workflow building blocks (AND-split, AND-join, OR-split, and OR-join) are supported. Second, WF-nets that are not SWF-nets are typically difficult to understand and should be avoided, if possible. Third, many workflow management systems only allow for workflow processes that correspond to SWF-nets. The latter observation can be explained by the fact that most workflow management systems use a language with separate building blocks for OR-splits and AND-joins. Finally, there is a very pragmatic argument. If we drop any of the requirements stated in Definition 4.2, relation $>_W$ does not contain enough information to successfully mine all processes in the resulting class.

The reader familiar with Petri nets will observe that SWF-nets belong to the class of free-choice nets [12]. This allows us to use efficient analysis techniques and advanced theoretical results. For example, using these results, it is possible to decide soundness in polynomial time [2].

SWF-nets also satisfy another interesting property.

Property 4.1. *Let $N = (P, T, F)$ be an SWF-net. For any $a, b \in T$ and $p_1, p_2 \in P$: if $p_1 \in a \bullet \cap \bullet b$ and $p_2 \in a \bullet \cap \bullet b$, then $p_1 = p_2$.*

This property follows directly from the definition of SWF-nets and states that no two transitions are connected by multiple places. This property illustrates that the structure of an SWF-net clearly reflects its behavior and

vice versa. This is exactly what we need to be able to rediscover a WF-net from its log.

We already showed that causal relations in $\rightarrow_W$ imply the presence of places. Now, we try to prove the reverse for the class of SWF-nets. First, we focus on the relation between the presence of places and $>_W$.

Theorem 4.2. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete workflow log of N. For any*

$$a, b \in T : a \bullet \cap \bullet b \neq \emptyset$$

implies $a >_W b$.

Proof. See [6]. □

Unfortunately, $a \bullet \cap \bullet b \neq \emptyset$ does not imply $a \rightarrow_W b$. To illustrate this, consider Fig. 4. For the first two nets (i.e., N_1 and N_2), two tasks are connected iff there is a causal relation. This does not hold for N_3 and N_4. In N_3, $A \rightarrow_{W_3} B$, $A \rightarrow_{W_3} D$, and $B \rightarrow_{W_3} D$. However, not $B \rightarrow_{W_3} B$. Nevertheless, there is a place connecting B to B. In N_4, although there are places connecting B to C and vice versa, $B \nrightarrow_{W_3} C$ and $B \nrightarrow_{W_3} C$. These examples indicate that loops of length one (see N_3) and length two (see N_4) are harmful. Fortunately, loops of length three or longer are no problem as is illustrated in the following theorem.

Theorem 4.3. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete workflow log of N. For any $a, b \in T$: $a \bullet \cap \bullet b \neq \emptyset$ and $b \bullet \cap \bullet a = \emptyset$ implies $a \rightarrow_W b$.*

Proof. See [6]. □

Acyclic nets have no loops of length one or length two. Therefore, it is easy to derive the following property.

Property 4.2. *Let $N = (P, T, F)$ be an acyclic sound SWF-net and let W be a complete workflow log of N. For any $a, b \in T$: $a \bullet \cap \bullet b \neq \emptyset$ iff $a \rightarrow_W b$.*

The results presented thus far focus on the correspondence between connecting places and causal relations. However, causality ($\rightarrow_W$) is just one of the four log-based ordering relations defined in Definition 4.2. The following theorem explores the relation between the sharing of input and output places and $\#_W$.

Theorem 4.4. *Let $N = (P, T, F)$ be a sound SWF-net such that for any $a, b \in T$: $a \bullet \cap \bullet b = \emptyset$ or $b \bullet \cap \bullet a = \emptyset$ and let W be a complete workflow log of N.*

1. *If $a, b \in T$ and $a \bullet \cap b \bullet \neq \emptyset$, then $a \#_W b$.*
2. *If $a, b \in T$ and $\bullet a \cap \bullet b \neq \emptyset$, then $a \#_W b$.*
3. *If $a, b, t \in T$, $a \rightarrow_W t$, $b \rightarrow_W t$, and $a \#_W b$, then $a \bullet \cap b \bullet \cap \bullet t \neq \emptyset$.*
4. *If $a, b, t \in T$, $t \rightarrow_W a$, $t \rightarrow_W b$, and $a \#_W b$, then $\bullet a \cap \bullet b \cap t \bullet \neq \emptyset$.*

Proof. See [6]. □

The relations $\rightarrow_W$, $\rightarrow_W^{-1}$, $\#_W$, and $\|_W$ are mutually exclusive. Therefore, we can derive that for sound SWF-nets with no short loops, $a \|_W b$ implies $a \bullet \cap b \bullet = \bullet a \cap \bullet b = \emptyset$. Moreover, $a \rightarrow_W t$, $b \rightarrow_W t$, and $a \bullet \cap b \bullet \cap \bullet t = \emptyset$ implies $a \|_W b$. Similarly, $t \rightarrow_W a$, $t \rightarrow_W b$, and $\bullet a \cap \bullet b \cap t \bullet = \emptyset$, also

implies $a \|_W b$. These results will be used to underpin the mining algorithm presented in the following section.

4.3 Mining Algorithm

Based on the results in the previous sections, we now present an algorithm for mining processes. The algorithm uses the fact that for many WF-nets, two tasks are connected iff their causality can be detected by inspecting the log.

Definition 4.3 (Mining algorithm α). *Let W be a workflow log over T. $\alpha(W)$ is defined as follows:*

1. $T_W = \{t \in T \mid \exists_{\sigma \in W} t \in \sigma\}$,
2. $T_I = \{t \in T \mid \exists_{\sigma \in W} t = first(\sigma)\}$,
3. $T_O = \{t \in T \mid \exists_{\sigma \in W} t = last(\sigma)\}$,
4.
$$X_W = \{(A, B) \mid A \subseteq T_W \wedge B \subseteq T_W$$
$$\wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2$$
$$\wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\},$$

5.
$$Y_W = \{(A, B) \in X_W \mid \forall_{(A', B') \in X_W} A \subseteq A'$$
$$\wedge B \subseteq B' \Longrightarrow (A, B) = (A', B')\},$$

6. $P_W = \{p_{(A,B)} \mid (A, B) \in Y_W\} \cup \{i_W, o_W\}$,
7.
$$F_W = \{(a, p_{(A,B)}) \mid (A, B) \in Y_W \wedge a \in A\}$$
$$\cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_W \wedge b \in B\}$$
$$\cup \{(i_W, t) \mid t \in T_I\} \cup \{(t, o_W) \mid t \in T_O\},$$

and
8. $\alpha(W) = (P_W, T_W, F_W)$.

The mining algorithm constructs a net (P_W, T_W, F_W). Clearly, the set of transitions T_W can be derived by inspecting the log. In fact, as shown in Property 3.2, if there are no traces of length one, T_W can be derived from $>_W$. Since it is possible to find all initial transitions T_I and all final transition T_O, it is easy to construct the connections between these transitions and i_W and o_W. Besides the source place i_W and the sink place o_W, places of the form $p_{(A,B)}$ are added. For such place, the subscript refers to the set of input and output transitions, i.e., $\bullet p_{(A,B)} = A$ and $p_{(A,B)} \bullet = B$. A place is added in-between a and b iff $a \rightarrow_W b$. However, some of these places need to be merged in case of OR-splits/joins rather than AND-splits/joins. For this purpose, the relations X_W and Y_W are constructed. $(A, B) \in X_W$ if there is a causal relation from each member of A to each member of B and the members of A and B never occur next to one another. Note that, if $a \rightarrow_W b$, $b \rightarrow_W a$, or $a \|_W b$, then a and b cannot be both in A (or B). Relation Y_W is derived from X_W by taking only the largest elements with respect to set inclusion. (See the end of this section for an example.)

Based on α defined in Definition 4.3, we turn to the rediscovery problem. Is it possible to rediscover WF-nets using $\alpha(W)$? Consider the five SWF-nets shown in Fig. 4. If
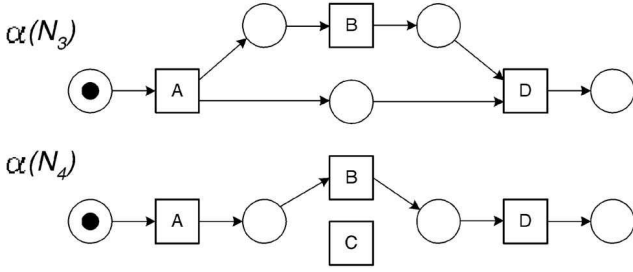
Fig. 5. The α algorithm is unable to rediscover N_3 and N_4.

α is applied to a complete workflow log of N_1, the resulting net is N_1 modulo renaming of places. Similarly, if α is applied to a complete workflow log of N_2, the resulting net is N_2 modulo renaming of places. As expected, α is not able to rediscover N_3 and N_4 (see Fig. 5). $\alpha(W_3)$ is like N_3, but without the arcs connecting B to the place in-between A and D and two new places. $\alpha(W_4)$ is like N_4, but the input and output arc of C are removed. $\alpha(W_3)$ is not a WF-net since B is not connected to the rest of the net. $\alpha(W_4)$ is not a WF-net since C is not connected to the rest of the net. In both cases, two arcs are missing in the resulting net. N_3 and N_4 illustrate that the mining algorithm is unable to deal with short loops. Loops of length three or longer are no problem. For example, $\alpha(W_5) = N_5$ modulo renaming of places. The following theorem proves that α is able to rediscover the class of SWF-nets provided that there are no short loops.

Theorem 4.4. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete workflow log of N. If for all $a, b \in T$ $a \bullet \cap \bullet b = \emptyset$ or $b \bullet \cap \bullet a = \emptyset$, then $\alpha(W) = N$ modulo renaming of places.*

Proof. Let $\alpha(W) = (P_W, T_W, F_W)$. Since W is complete, it is easy to see that $T = T_W$. It remains to be proven that every place in N corresponds to a place in $\alpha(W)$ and vice versa.

Let $p \in P$. We need to prove that there is a $p_W \in P_W$ such that $\overset{N}{\bullet}p = \overset{N_W}{\bullet}p_W$ and $p\overset{N}{\bullet} = p_W\overset{N_W}{\bullet}$. If $p = i$, i.e., the source place or $p = o$, i.e., the sink place, then it is easy to see that there is a corresponding place in $\alpha(W)$. Transitions in $i\overset{N}{\bullet} \cup \overset{N}{\bullet}o$ can fire only once directly at the beginning of a sequence or at the end. Therefore, the construction given in Definition 4.3 involving i_W, o_W, T_I, and T_O yields a source and sink place with identical input/output transitions. If $p \notin \{i, o\}$, then let $A = \overset{N}{\bullet}p$, $B = p\overset{N}{\bullet}$, and $p_W = p_{(A,B)}$. If p_W is indeed a place of $\alpha(W)$, then $\overset{N}{\bullet}p = \overset{\alpha(W)}{\bullet}p_W$ and $p\overset{N}{\bullet} = p_W\overset{\alpha(W)}{\bullet}$. This follows directly from the definition of the flow relation F_W in Definition 4.3. To prove that $p_W = p_{(A,B)}$ is a place of $\alpha(W)$, we need to show that $(A, B) \in Y_W$. $(A, B) \in X_W$ because

1. Theorem 4.3 implies that $\forall_{a \in A} \forall_{b \in B} a \to_W b$,
2. Theorem 4.4, item 1 implies that $\forall_{a_1, a_2 \in A} a_1 \#_W a_2$, and

3. Theorem 4.4, item 2 implies that $\forall_{b_1, b_2 \in B} b_1 \#_W b_2$.

To prove that $(A, B) \in Y_W$, we need to show that it is not possible to have $(A', B') \in X_W$ such that $A \subseteq A'$, $B \subseteq B'$, and $(A, B) \neq (A', B')$ (i.e., $A \subset A'$ or $B \subset B'$). Suppose that $A \subset A'$. There is an $a' \in T \setminus A$ such that $\forall_{b \in B} a' \to_W b$ and $\forall_{a \in A} a \#_W a'$. Theorem 4.4, item 3 implies that $a\overset{N}{\bullet} \cap a'\overset{N}{\bullet} \cap \overset{N}{\bullet}b \neq \emptyset$ for some $b \in B$. Let $p' \in a\overset{N}{\bullet} \cap a'\overset{N}{\bullet} \cap \overset{N}{\bullet}b$. Property 4.1 implies $p' = p$. However, $a' \notin A = \overset{N}{\bullet}p$ and $a' \in \overset{N}{\bullet}p'$, and we find a contradiction ($p' = p$ and $p' \neq p$). Suppose that $B \subset B'$. There is a $b' \in T \setminus B$ such that $\forall_{a \in A} a \to_W b'$ and $\forall_{b \in B} b \#_W b'$. Using Theorem 4.4, item 4 and Property 4.1, we can show that this leads to a contradiction. Therefore, $(A, B) \in Y_W$ and $p_W \in P_W$.

Let $p_w \in P_W$. We need to prove that there is a $p \in P$ such that $\overset{N}{\bullet}p = \overset{N_W}{\bullet}p_W$ and $p\overset{N}{\bullet} = p_W\overset{N_W}{\bullet}$. If $p_w = i_w$ or $p_w = o_w$, then p_w corresponds to i or o, respectively. This is a direct consequence of the construction given in Definition 4.3 involving i_W, o_W, T_I, and T_O. If $p_w \notin \{i_w, o_w\}$, then there are sets A and B such that $(A, B) \in Y_W$ and $p_w = p_{(A,B)}$. $\overset{\alpha(N)}{\bullet}p_w = A$ and $p_w\overset{\alpha(N)}{\bullet} = B$. It remains to be proven that there is a $p \in P$ such that $\overset{N}{\bullet}p = A$ and $p\overset{N}{\bullet} = B$. Since $(A, B) \in Y_W$ implies that $(A, B) \in X_W$, for any $a \in A$ and $b \in B$ there is a place connecting a and b (use $a \to_W b$ and Theorem 4.1). Using Theorem 4.4, we can prove that there is just one such place. Let p be this place. Clearly, $\overset{N}{\bullet}p \subseteq A$ and $p\overset{N}{\bullet} \subseteq B$. It remains to be proven that $\overset{N}{\bullet}p = A$ and $p\overset{N}{\bullet} = B$. Suppose that $a' \in \overset{N}{\bullet}p \setminus A$ (i.e., $\overset{N}{\bullet}p \neq A$). Select an arbitrary $a \in A$ and $b \in B$. Using Theorem 4.3, we can show that $a' \to_W b$. Using Theorem 4.4, item 1, we can show that $a \#_W a'$. This holds for any $a \in A$ and $b \in B$. Therefore, $(A \cup \{a'\}, B) \in X_W$. However, this is not possible since $(A, B) \in Y_W$ ((A, B) should be maximal). Therefore, we find a contradiction. We find a similar contradiction if we assume that there is a $b' \in p\overset{N}{\bullet} \setminus B$. Therefore, we conclude that $\overset{N}{\bullet}p = A$ and $p\overset{N}{\bullet} = B$. □

Nets N_1, N_2, and N_5 shown in Fig. 4 satisfy the requirements stated in Theorem 4.4. Therefore, it is no surprise that α is able to rediscover these nets. The net shown in Fig. 1 is also an SWF-net with no short loops. Therefore, we can successfully rediscover the net if the AND-split and the AND-join are *visible in the log*. The latter assumption is not realistic if these two transitions do not correspond to real work. Given the fact the log shown in Table 1 does not list the occurrence of these events, indicates that this assumption is not valid. Therefore, the AND-split and the AND-join should be considered invisible. However, if we apply α to this log

$$W = \{ABCD, ACBD, AED\},$$

then the result is quite surprising. The resulting net $\alpha(W)$ is shown in Fig. 6.
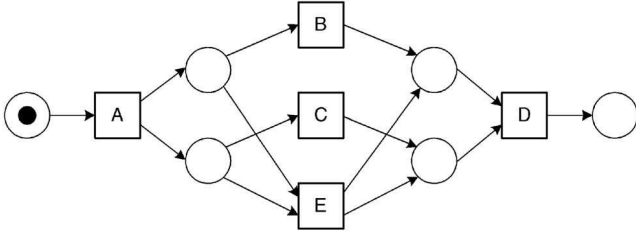
Fig. 6. Another process model corresponding to the workflow log shown in Table 1.

To illustrate the α algorithm we show the result of each step using the log $W = \{ABCD, ACBD, AED\}$ (i.e., a log like the one shown in Table 1):

1. $T_W = \{A, B, C, D, E\}$,
2. $T_I = \{A\}$,
3. $T_O = \{D\}$,
4.

$$X_W = \{(\{A\}, \{B\}), (\{A\}, \{C\}), (\{A\}, \{E\}),$$
$$(\{B\}, \{D\}), (\{C\}, \{D\}), (\{E\}, \{D\}),$$
$$(\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}),$$
$$(\{C, E\}, \{D\})\},$$

5.

$$Y_W = \{(\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\},$$
$$\{D\}), (\{C, E\}, \{D\})\},$$

6.

$$P_W = \{i_W, o_W, p_{(\{A\}, \{B, E\})}, p_{\{A\}, \{C, E\}},$$
$$p_{(\{B, E\}, \{D\})}, p_{(\{C, E\}, \{D\})}\},$$

7.

$$F_W = \{(i_W, A), (A, p_{(\{A\}, \{B, E\})}),$$
$$(p_{(\{A\}, \{B, E\})}, B) \ldots, (D, o_W)\},$$

and

8. $\alpha(W) = (P_W, T_W, F_W)$ (as shown in Fig. 6).

Although the resulting net is not an SWF-net, it is a sound WF-net whose observable behavior is identical to the net shown in Fig. 1. Also note that the WF-net shown in Fig. 6 can be rediscovered, although it is not an SWF-net. This example shows that the applicability is not limited to SWF-nets. However, for arbitrary sound WF-nets, it is not possible to guarantee that they can be rediscovered.

4.4 Limitations of the α Algorithm

As demonstrated through Theorem 4.4, the α algorithm is able to rediscover a large class of processes. However, we did not prove that the class of processes is maximal, i.e., that there is not a "better" algorithm able to rediscover even more processes. Therefore, we reflect on the requirements stated in Definition 4.2 (SWF-nets) and Theorem 4.4 (no short loops).
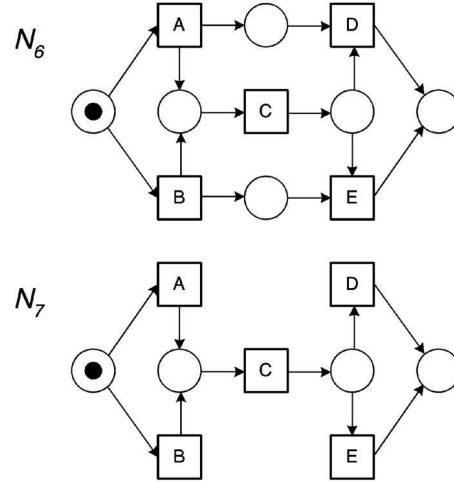

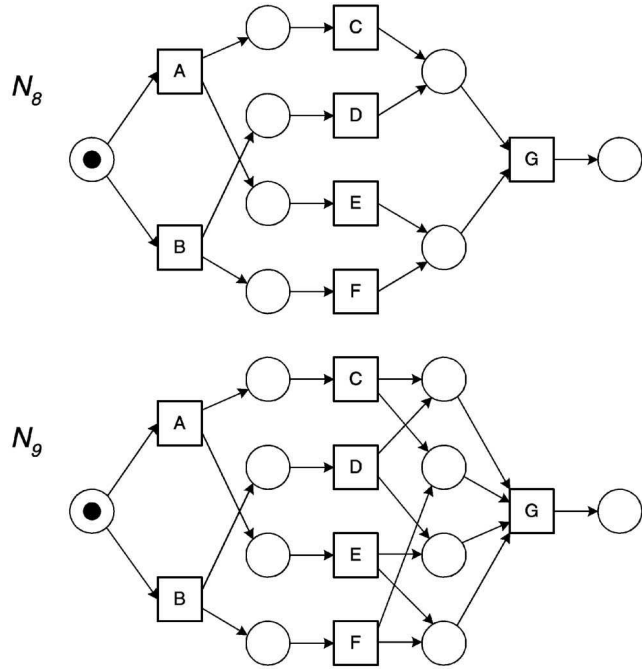Fig. 7. The nonfree-choice WF-net N_6 cannot be rediscovered by the α algorithm.

Fig. 8. WF-net N_8 cannot be rediscovered by the α algorithm. Nevertheless, α returns a WF-net which is behavioral equivalent.

Let us first consider the requirements stated in Definition 4.2. To illustrate the necessity of the first two requirements consider Figs. 7 and 8. The WF-net N_6 shown in Fig. 7 is sound, but not an SWF-net since the first requirement is violated (N_6 is not free-choice). If we apply the mining algorithm to a complete workflow log W_6 of N_6, we obtain the WF-net N_7 also shown in Fig. 7 (i.e., $\alpha(W_6) = N_7$). Clearly, N_6 cannot be rediscovered using α. Although N_7 is a sound SWF-net, its behavior is different from N_6, e.g., workflow trace ACE is possible in N_7 but not in N_6. This example motivates the first requirement in Definition 4.2. The second requirement is motivated by Fig. 8. N_8 violates the second requirement. If we apply the mining algorithm to a complete workflow log W_8 of N_8, we obtain the WF-net $\alpha(W_8) = N_9$ also shown in Fig. 8.
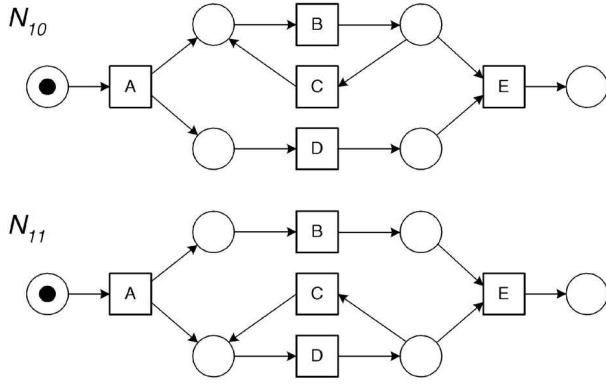
Fig. 9. Although both WF-nets are not behavioral equivalent they are identical with respect to $>$.

Fig. 10. Both SWF-nets are behavioral equivalent and, therefore, any algorithm will be unable to distinguish N_{12} from N_{13} (assuming a notion of completeness based on $>$).

Although N_9 is behaviorally equivalent, N_8 cannot be rediscovered using the mining algorithm.

Although the requirements stated in Definition 4.2 are necessary in order to prove that this class of workflow processes can be rediscovered on the basis of a complete workflow log, the applicability is not limited to SWF-nets. The examples given in this section show that in many situations, a behaviorally equivalent WF-net can be derived. Note that the third requirement stated in Definition 4.2 (no implicit places) can be dropped thus allowing even more behaviorally equivalent WF-nets. Even in the cases where the resulting WF-net is not behaviorally equivalent, the results are meaningful, e.g., the process represented by N_7 is different from the process represented by N_6 (cf. Fig. 7). Nevertheless, N_7 is similar and captures most of the behavior.

Another requirement imposed by Theorem 4.4 is the absence of short loops. We already showed examples motivating this requirement. However, it is clear that the α algorithm can be improved to deal with short loops. Unfortunately, this is less trivial than it may seem. We use Fig. 9 and Fig. 10 to illustrate this. Fig. 9 shows two WF-nets. Although both WF-nets have clearly different behaviors, their complete logs will have identical $>$ relations. Note that the two WF-nets are not SWF-nets because of the nonfree-choice construct involving E. However, the essence of the problem is the short loop involving C. Because of this short loop, the α algorithm will create for both N_{10} and N_{11} the Petri net where C is unconnected to the rest of the net. Clearly, this can be improved since for complete logs of N_{10} and N_{11} the following relations hold: $B > C$, $C > B$, $C > D$, $D > C$, $B\|C$, $C\|D$, and $B\|D$. These relations suggest that B, C, and D are in parallel. However, in any complete log, it can be seen that this is not the case since A is never followed by C. Despite this information, no algorithm will be able to distinguish N_{10} and N_{11} because they are identical with respect to $>$. Fig. 10 gives another example demonstrating that dealing with short loops is far from trivial. N_{12} and N_{13} are SWF-nets that are behavioral equivalent, therefore, no algorithm will be able to distinguish N_{12} from N_{13} (assuming the notion of completeness and not logging explicit start and end events). This problem may seem similar to Fig. 8 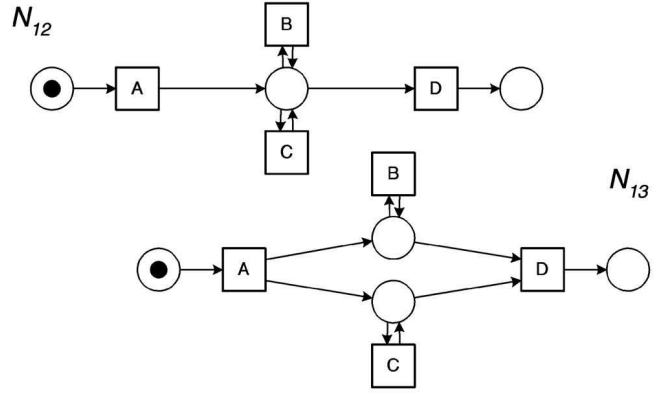or examples with and without implicit places. However, N_{12} and N_{13} are SWF-nets, while N_8 or examples with implicit places are just WF-nets not satisfying the requirements stated in Definition 4.2.

Despite the problems illustrated by Fig. 9 and Fig. 10, it is possible to solve the problem using a slightly stronger notion of completeness. Assume that it is possible to identify 1-loops and 2-loops. This can be done by looking for patterns AA (1-loop) and ABA (2-loop). Then, refine relation $>$ into a new relation $>^+$ which splits the existing transitions involved in a short loop into two or three transitions. Tasks involved in a 1-loop are mapped onto three transitions (e.g., start, execute, and end). Tasks involved in a 2-loop, but not a 1-loop are mapped onto two transitions (e.g., start and end). The goal of this translation is to make the short loops longer by using additional information about 1-loops and 2-loops (and while preserving the original properties). This refined relation $>^+$ can be used as input for the α algorithm. This approach is able to deal with all possible situations except the one illustrated in Fig. 10. Note that no algorithm will be able to distinguish the logs of the two processes shown in Fig. 10. However, this is not a real problem since they are behaviorally equivalent. In formal terms, we can replace the requirement in Theorem 4.4 by the weaker requirement that there are no two transitions having identical input and output places. A detailed description of this preprocessing algorithm and a proof of its correctness are, however, beyond the scope of this paper.

Besides the explicit requirements stated in Definition 4.2 and Theorem 4.4, there are also a number of implicit requirements. As indicated before, hidden tasks cannot be detected (cf. the AND-split and AND-join in Fig. 1). Moreover, our definition of a Petri net assumes that each transition bears a unique label. Instead, we could have used a labeled Petri net [39]. The latter choice would have been more realistic but also complicate matters enormously. The current definition of a WF-net assumes that each task appears only once in the network. Without this assumption, every occurrence of some task t could refer to one of multiple indistinguishable transitions with label t. Preliminary investigations show that the problems of "hidden tasks" and "duplicate tasks" are highly related, e.g., a WF-net with two transitions having the same label t has a corresponding WF-net with hidden transitions but only one transition labeled t. Moreover, there are relations between

the explicit requirements stated in Definition 4.2 and Theorem 4.4 and the implicit requirements just mentioned, e.g., in some cases, a nonfree-choice WF-net can be made free choice by inserting hidden transitions. These findings indicate that the class of SWF-nets is close to the upper bound of workflow processes that can be mined successfully given the notion of completeness stated in Definition 3.4. To move beyond SWF-nets, we either have to resort to heuristics or strengthen the notion of completeness (and thus require more observations).

To conclude this section, we consider the complexity of the α algorithm. Event logs may be huge containing millions of events. Fortunately, the α algorithm is driven by relation $>$. The time it takes to build this relation is linear in the size of the log. The complexity of the remaining steps in α algorithm is exponential in the number of tasks. However, note that the number of tasks is typically less than 100 and does not depend on the size of the log. Therefore, the complexity is not a bottleneck for large-scale application.

5 RELATED WORK

The idea of process mining is not new [7], [9], [10], [11], [19], [20], [21], [22], [23], [24], [31], [32], [33], [41], [42], [43], [44], [45], [48], [49], [50]. Cook and Wolf have investigated similar issues in the context of software engineering processes. In [9], they describe three methods for process discovery: one using neural networks, one using a purely algorithmic approach, and one Markovian approach. The authors consider the latter two the most promising approaches. The purely algorithmic approach builds a finite state machine where states are fused if their futures (in terms of possible behavior in the next k steps) are identical. The Markovian approach uses a mixture of algorithmic and statistical methods and is able to deal with noise. Note that the results presented in [9] are limited to sequential behavior. Related, but in a different domain, is the work presented in [29], [30], also using a Markovian approach restricted to sequential processes. Cook and Wolf extend their work to concurrent processes in [10]. They propose specific metrics (entropy, event type counts, periodicity, and causality) and use these metrics to discover models out of event streams. However, they do not provide an approach to generate explicit process models. Recall that the final goal of the approach presented in this paper is to find explicit representations for a broad range of process models, i.e., we want to be able to generate a concrete Petri net rather than a set of dependency relations between events. In [11], Cook and Wolf provide a measure to quantify discrepancies between a process model and the actual behavior as registered using event-based data. The idea of applying process mining in the context of workflow management was first introduced in [7]. This work is based on workflow graphs, which are inspired by workflow products such as IBM MQSeries workflow (formerly known as Flowmark) and InConcert. In this paper, two problems are defined. The first problem is to find a workflow graph generating events appearing in a given workflow log. The second problem is to find the definitions of edge conditions. A concrete algorithm is given for tackling the first problem. The approach is quite different from other approaches: Because of the nature of workflow graphs, there is no need

to identify the nature (AND or OR) of joins and splits. As shown in [27], workflow graphs use true and false tokens which do not allow for cyclic graphs. Nevertheless, [7] partially deals with iteration by enumerating all occurrences of a given task and then folding the graph. However, the resulting conformal graph is not a complete model. In [33], a tool based on these algorithms is presented. Schimm [41], [42], [45] has developed a mining tool suitable for discovering hierarchically structured workflow processes. This requires all splits and joins to be balanced. Herbst also address the issue of process mining in the context of workflow management [21], [19], [20], [23], [24], [22] using an inductive approach. The work presented in [22], [24] is limited to sequential models. The approach described in [21], [19], [20], [23] also allows for concurrency. It uses stochastic task graphs as an intermediate representation and it generates a workflow model described in the ADONIS modeling language. In the induction step, task nodes are merged and split in order to discover the underlying process. A notable difference with other approaches is that the same task can appear multiple times in the workflow model, i.e., the approach allows for duplicate tasks. The graph generation technique is similar to the approach of [7], [33]. The nature of splits and joins (i.e., AND or OR) is discovered in the transformation step, where the stochastic task graph is transformed into an ADONIS workflow model with block-structured splits and joins. In contrast to the previous papers, our work [31], [32], [48], [49], [50] is characterized by the focus on workflow processes with concurrent behavior (rather than adding ad hoc mechanisms to capture parallelism). In [48], [49], [50], a heuristic approach using rather simple metrics is used to construct so-called "dependency/frequency tables" and "dependency/frequency graphs". In [31], another variant of this technique is presented using examples from the health-care domain. The preliminary results presented in [31], [48], [49], [50] only provide heuristics and focus on issues such as noise. The approach described in this paper differs from these approaches in the sense that for the α algorithm it is proven that for certain subclasses, it is possible to find the right workflow model. In [4], the EMiT tool is presented which uses an extended version of α algorithm to incorporate timing information. Note that in this paper, there is no detailed description of the α algorithm nor a proof of its correctness.

Process mining can be seen as a tool in the context of Business (Process) Intelligence (BPI). In [18], [40], a BPI toolset on top of HP's Process Manager is described. The BPI tools set includes a so-called "BPI Process Mining Engine." However, this engine does not provide any techniques as discussed before. Instead, it uses generic mining tools such as SAS Enterprise Miner for the generation of decision trees relating attributes of cases to information about execution paths (e.g., duration). In order to do workflow mining it is convenient to have a so-called "process data warehouse" to store audit trails. Such a data warehouse simplifies and speeds up the queries needed to derive causal relations. In [13], [34], [35], [36], the design of such warehouse and related issues are discussed in the context of workflow logs. Moreover, [36] describes the PISA tool which can be used to extract performance metrics from workflow logs. Similar diagnostics are provided by the ARIS Process Performance Manager (PPM) [25]. The later tool is commercially available

and a customized version of PPM is the Staffware Process Monitor (SPM) [46] which is tailored toward mining Staffware logs. Note that the goal of the latter tools is not to extract the process model. The main focus is on clustering and performance analysis rather than causal relations as in [7], [9], [10], [11], [19], [20], [21], [22], [23], [24], [31], [32], [33], [41], [42], [43], [44], [45], [48], [49], [50].

More from a theoretical point of view, the rediscovery problem discussed in this paper is related to the work discussed in [8], [16], [17], [38]. In these papers, the limits of inductive inference are explored. For example, in [17], it is shown that the computational problem of finding a minimum finite-state acceptor compatible with given data is NP-hard. Several of the more generic concepts discussed in these papers could be translated to the domain of process mining. It is possible to interpret the problem described in this paper as an inductive inference problem specified in terms of rules, a hypothesis space, examples, and criteria for successful inference. The comparison with literature in this domain raises interesting questions for process mining, e.g., how to deal with negative examples (i.e., suppose that besides log W, there is a log V of traces that are not possible, e.g., added by a domain expert). However, despite the many relations with the work described in [8], [16], [17], [38], there are also many differences, e.g., we are mining at the net level rather than sequential or lower-level representations (e.g., Markov chains, finite state machines, or regular expressions).

Additional related work is the seminal work on regions [14]. This work investigates which transition systems can be represented by (compact) Petri nets (i.e., the so-called synthesis problem). Although the setting is different and our notion of completeness is much weaker than actually knowing the transition system, there are related problems such as duplicate transitions, etc.

6   CONCLUSION

In this paper, we addressed the workflow rediscovery problem. This problem was formulated as follows: "Find a mining algorithm able to rediscover a large class of sound WF-nets on the basis of complete workflow logs." We presented the α algorithm that is able to rediscover a large and relevant class of workflow processes (SWF-nets). Through examples, we also showed that the algorithm provides interesting analysis results for workflow processes outside this class. At this point in time, we are improving the mining algorithm such that it is able to rediscover an even larger class of WF-nets. We have tackled the problem of short loops and are now focusing on hidden tasks, duplicate tasks, and advanced routing constructs. However, given the observation that the class of SWF-nets is close to the upper limit of what one can do assuming this notion of completeness, new results will either provide heuristics or require stronger notions of completeness (i.e., more observations).

It is important to see the results presented in this paper in the context of a larger effort [31], [32], [48], [49], [50]. The rediscovery problem is not a goal by itself. The overall goal is to be able to analyze any workflow log without any knowledge of the underlying process and in the presence of noise. The theoretical results presented in this paper provide insights that are consistent with empirical results found earlier [31], [32], [48], [49], [50]. It is quite interesting

to see that the challenges encountered in practice match the challenges encountered in theory. For example, the fact that workflow process exhibiting nonfree-choice behavior (i.e., violating the first requirement of Definition 4.2) are difficult to mine was observed both in theory and in practice. Therefore, we consider the work presented in this paper as a stepping stone for good and robust process mining techniques.

We have applied our workflow mining techniques to two real applications. The first application is in health-care where the flow of multidisciplinary patients is analyzed. We have analyzed workflow logs (visits to different specialists) of patients with peripheral arterial vascular diseases of the Elizabeth Hospital in Tilburg and the Academic Hospital in Maastricht. Patients with peripheral arterial vascular diseases are a typical example of multidisciplinary patients. We have preliminary results showing that process mining is very difficult given the "spaghetti-like" nature of this process. Only by focusing on specific tasks and abstracting from infrequent tasks are we able to successfully mine such processes. The second application concerns the processing of fines by the CJIB (Centraal Justitieel Incasso Bureau), the Dutch Judicial Collection Agency located in Leeuwarden. We have successfully mined the process using information from 130,136 cases. The process comprises 99 tasks and has been validated by the CJIB. This positive result shows that process mining based on the α algorithm and using tools like EMiT and Little Thumb is feasible for at least structured processes. These findings are encouraging and show the potential of the α algorithm presented in this paper.
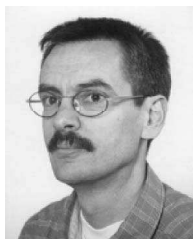
REFERENCES

[1]   W.M.P. van der Aalst, "Verification of Workflow Nets," *Application and Theory of Petri Nets,* P. Azema and G. Balbo, eds., pp. 407-426, Berlin: Springer-Verlag, 1997.

[2]   W.M.P. van der Aalst, "The Application of Petri Nets to Workflow Management," *The J. Circuits, Systems and Computers,* vol. 8, no. 1, pp. 21-66, 1998.

[3]   "Business Process Management: Models, Techniques, and Empirical Studies," *Lecture Notes in Computer Science,* W.M.P. van der Aalst, J. Desel, and A. Oberweis, eds., vol. 1806, Springer-Verlag, Berlin, 2000.

[4]   W.M.P. van der Aalst and B.F. van Dongen, "Discovering Workflow Performance Models from Timed Logs," *Proc. Int'l Conf. Eng. and Deployment of Cooperative Information Systems (EDCIS 2002),* Y. Han, S. Tai, and D. Wikarski, eds., vol. 2480, pp. 45-63, 2002.

[5]   W.M.P. van der Aalst and K.M. van Hee, *Workflow Management: Models, Methods, and Systems.* Cambridge, Mass.: MIT Press, 2002.

[6]   W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster, "Workflow Mining: Which Processes can be Rediscovered?" BETA Working Paper Series, WP 74, Eindhoven Univ. of Technology, Eindhoven, 2002.

[7]   R. Agrawal, D. Gunopulos, and F. Leymann, "Mining Process Models from Workflow Logs," *Proc. Sixth Int'l Conf. Extending Database Technology,* pp. 469-483, 1998.

[8]   D. Angluin and C.H. Smith, "Inductive Inference: Theory and Methods," *Computing Surveys,* vol. 15, no. 3, pp. 237-269, 1983.

[9]   J.E. Cook and A.L. Wolf, "Discovering Models of Software Processes from Event-Based Data," *ACM Trans. Software Eng. and Methodology,* vol. 7, no. 3, pp. 215-249, 1998.

[10] J.E. Cook and A.L. Wolf, "Event-Based Detection of Concurrency," *Proc. Sixth Int'l Symp. the Foundations of Software Eng. (FSE-6),* pp. 35-45, 1998.

[11] J.E. Cook and A.L. Wolf, "Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model," *ACM Trans. Software Eng. and Methodology,* vol. 8, no. 2, pp. 147-176, 1999.

[12] J. Desel and J. Esparza, "Free Choice Petri Nets," *Cambridge Tracts in Theoretical Computer Science,* vol. 40, Cambridge, UK: Cambridge Univ. Press, 1995.

[13] J. Eder and G.E. Olivotto, and W. Gruber, "A Data Warehouse for Workflow Logs," *Proc. Int'l Conf. Eng. and Deployment of Cooperative Information Systems (EDCIS 2002),* Y. Han, S. Tai, and D. Wikarski, eds., pp. 1-15, 2002.

[14] A. Ehrenfeucht, G. Rozenberg, "Partial (Set) 2-Structures—Part 1 and Part 2," *Acta Informatica,* vol. 27, no. 4, pp. 315-368, 1989.

[15] *Workflow Handbook 2001, Workflow Management Coalition,* L. Fischer, ed. Lighthouse Point, Fla.: Future Strategies, 2001.

[16] E.M. Gold, "Language Identfication in the Limit," *Information and Control,* vol. 10, no. 5, pp. 447-474, 1967.

[17] E.M. Gold, "Complexity of Automaton Identification from Given Data," *Information and Control,* vol. 37, no. 3, pp. 302-320, 1978.

[18] D. Grigori, F. Casati, U. Dayal, and M.C. Shan, "Improving Business Process Quality through Exception Understanding, Prediction, and Prevention," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01),* P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, eds., pp. 159-168, 2001.

[19] J. Herbst, "A Machine Learning Approach to Workflow Management," *Proc. 11th European Conf. Machine Learning,* pp. 183-194, 2000.

[20] J. Herbst, "Dealing with Concurrency in Workflow Induction," *Proc. European Concurrent Eng. Conf.,* U. Baake, R. Zobel, and M. Al-Akaidi, eds., 2000.

[21] J. Herbst, "Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen," PhD thesis, Universität Ulm, Nov. 2001.

[22] J. Herbst and D. Karagiannis, "Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models," *Proc. Ninth Int'l Workshop Database and Expert Systems Applications,* pp. 745-752, 1998.

[23] J. Herbst and D. Karagiannis, "An Inductive Approach to the Acquisition and Adaptation of Workflow Models," *Proc. Workshop Intelligent Workflow and Process Management: The New Frontier for AI in Business,* M. Ibrahim and B. Drabble, eds., pp. 52-57, Aug. 1999.

[24] J. Herbst and D. Karagiannis, "Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models," *Int'l J. Intelligent Systems in Accounting, Finance, and Management,* vol. 9, pp. 67-92, 2000.

[25] IDS Scheer, ARIS Process Performance Manager (ARIS PPM), http://www.ids-scheer.com, 2002.

[26] S. Jablonski and C. Bussler, *Workflow Management: Modeling Concepts, Architecture, and Implementation.* London: Int'l Thomson Computer Press, 1996.

[27] B. Kiepuszewski, "Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows," PhD thesis, Queensland Univ. of Technology, Brisbane, Australia, 2002, available via http://www.tm.tue.nl/it/research/patterns.

[28] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques.* Upper Saddle River, New Jersey, Prentice-Hall PTR, 1999.

[29] H. Mannila and D. Rusakov, "Decomposing Event Sequences into Independent Components," *Proc. First SIAM Conf. Data Mining,* V. Kumar and R. Grossman, eds., pp. 1-17, 2001.

[30] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovery of Frequent Episodes in Event Sequences," *Data Mining and Knowledge Discovery,* vol. 1, no. 3, pp. 259-289, 1997.

[31] L. Maruster, W.M.P. van der Aalst, A.J.M.M. Weijters, A. van den Bosch, and W. Daelemans, "Automated Discovery of Workflow Models from Hospital Data," *Proc. 13th Belgium-Netherlands Conf. Artificial Intelligence (BNAIC 2001),* B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, eds., pp. 183-190, 2001.

[32] L. Maruster, A.J.M.M. Weijters, W.M.P. van der Aalst, and A. van den Bosch, "Process Mining: Discovering Direct Successors in Process Logs," *Proc. Fifth Int'l Conf. Discovery Science (Discovery Science 2002),* pp. 364-373, 2002.

[33] M.K. Maxeiner and K. Küspert, and F. Leymann, "Data Mining von Workflow-Protokollen zur teilautomatisierten Konstruktion von Prozessmodellen," *Proc. Datenbanksysteme in Büro, Technik und Wissenschaft,* pp. 75-84, 2001.

[34] M. zur Mühlen, "Process-Driven Management Information Systems—Combining Data Warehouses and Workflow Technology," *Proc. Int'l Conf. Electronic Commerce Research (ICECR-4),* B. Gavish, ed., pp. 550-566, 2001.

[35] M. zur Mühlen, "Workflow-Based Process Controlling-Or: What You Can Measure You Can Control," *Workflow Handbook 2001, Workflow Management Coalition,* L. Fischer, ed., pp. 61-77, Lighthouse Point, Fla.: Future Strategies, 2001.

[36] M. zur Mühlen and M. Rosemann, "Workflow-Based Process Monitoring and Controlling—Technical and Organizational Issues," *Proc. 33rd Hawaii Int'l Conf. System Science (HICSS-33),* R. Sprague, ed., pp. 1-10, 2000.

[37] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE,* vol. 77, no. 4, pp. 541-580, Apr. 1989.

[38] L. Pitt, "Inductive Inference, DFAs, and Computational Complexity," *Proc. Int'l Workshop Analogical and Inductive Inference (AII),* K.P. Jantke, ed., pp. 18-44, 1889.

[39] *Lectures on Petri Nets I: Basic Models,* Lecture Notes in Computer Science, vol. 1491, W. Reisig and G. Rozenberg, eds., Berlin: Springer-Verlag, 1998.

[40] M. Sayal, F. Casati, M.C. Shan, and U. Dayal, "Business Process Cockpit," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02),* pp. 880-883, 2002.

[41] G. Schimm, "Process Mining," http://www.processmining.de/, 2004.

[42] G. Schimm, "Generic Linear Business Process Modeling," *Proc. ER 2000 Workshop Conceptual Approaches for E-Business and The World Wide Web and Conceptual Modeling,* S.W. Liddle, H.C. Mayr, and B. Thalheim, eds., pp. 31-39, 2000.

[43] G. Schimm, "Process Mining Elektronischer Geschäftsprozesse," *Proc. Elektronische Geschäftsprozesse,* 2001.

[44] G. Schimm, "Process Mining linearer Prozessmodelle—Ein Ansatz zur Automatisierten Akquisition von Prozesswissen," *Proc. 1. Konferenz Professionelles Wissensmanagement,* 2001.

[45] G. Schimm, "Process Miner—A Tool for Mining Process Schemes from Event-Based Data," *Proc. Eighth European Conf. Artificial Intelligence (JELIA),* S. Flesca and G. Ianni, eds., pp. 525-528, 2002.

[46] Staffware, Staffware Process Monitor (SPM), http://www.staffware.com, 2002.

[47] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst, "Diagnosing Workflow Processes Using Woflan," *The Computer J.,* vol. 44, no. 4, pp. 246-279, 2001.

[48] A.J.M.M. Weijters and W.M.P. van der Aalst, "Process Mining: Discovering Workflow Models from Event-Based Data," *Proc. 13th Belgium-Netherlands Conf. Artificial Intelligence (BNAIC 2001),* B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, eds., pp. 283-290, 2001.

[49] A.J.M.M. Weijters and W.M.P. van der Aalst, "Rediscovering Workflow Models from Event-Based Data," *Proc. 11th Dutch-Belgian Conf. Machine Learning (Benelearn 2001),* V. Hoste and G. de Pauw, eds., pp. 93-100, 2001.

[50] A.J.M.M. Weijters and W.M.P. van der Aalst, "Workflow Mining: Discovering Workflow Models from Event-Based Data," *Proc. ECAI Workshop Knowledge Discovery and Spatial Data,* C. Dousson, F. Höppner, and R. Quiniou, eds., pp. 78-84, 2002.

Wil van der Aalst is a full professor of information systems and head of the section of information and technology in the Department of Technology Management at Eindhoven University of Technology. He is also a part-time full professor at the Computing Science section in the Department of Mathematics and Computer Science at the same university and an adjunct professor at Queensland University of Technology. His research interests include information systems, simulation, Petri nets, process models, workflow management systems, verification techniques, enterprise resource planning systems, computer supported cooperative work, and interorganizational business processes.

Ton Weijters is an associate professor in the Department of Technology Management at the Eindhoven University of Technology (TUE), and a member of the BETA research group. Currently, he is working on 1) the application of knowledge engineering and machine learning techniques for planning, scheduling, and process mining, and 2) fundamental research in the domain of machine learning and knowledge discovering. He is the author of many scientific publications in the mentioned research field.

Laura Maruster received the BS degree in 1994 and MS degree in 1995, both from the Computer Science Department at West University of Timisoara, Romania. At present, she is a PhD candidate in the Department of Technology Management at Eindhoven University of Technology, Eindhoven, The Netherlands. Her research interests include induction of machine learning and statistical models, process mining, and knowledge discovery.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.