Workflow Mining:
Which processes can be rediscovered?

W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster

Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
`w.m.p.v.d.aalst@tm.tue.nl`

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 (re)discovering workflow models. Starting point for such techniques are so-called "workflow logs" containing information about the workflow process as it is actually being executed. Unfortunately, it is not possible to (re)discover every workflow process. In this paper we explore the class of workflow processes which can be discovered. The theoretical results presented in this paper demonstrate that most practical workflow processes fit into this class. The tool *MiMo*, also presented in this paper, supports the (re)discovery of these processes.

Key words: Workflow mining, workflow management, data mining, Petri nets.

1 Introduction

During the last decade workflow management concepts and technology [4, 5, 11, 16, 17] 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 (i) each event refers to a task (i.e., a well-defined step in the workflow), (ii) each event refers to a case (i.e., a workflow instance),

and (iii) events are totally ordered. 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.

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

Table 1. A workflow log.

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 also task C is 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 Figure 1. The model is represented in terms of a Petri net [21]. 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 non-observable 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 for this example we assume that two tasks are in parallel if they appear in any order. By distinguishing between start events and end events for tasks it is possible to explicitly detect parallelism.
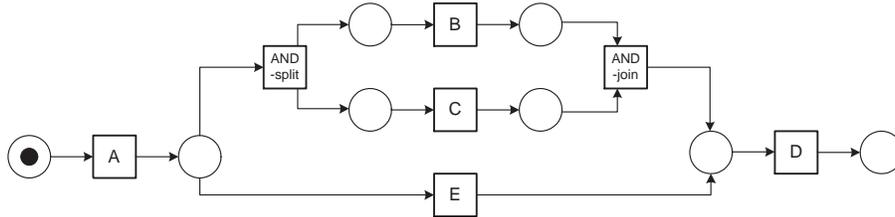

Fig. 1. A process model corresponding to the workflow log.

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 handing traffic violations: Is the make of a car relevant for the routing of the corresponding traffic violations? (E.g., 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 exceptions) 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 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 Figure 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 towards workflow processes. Figure 1 shows an example of a WF-net.
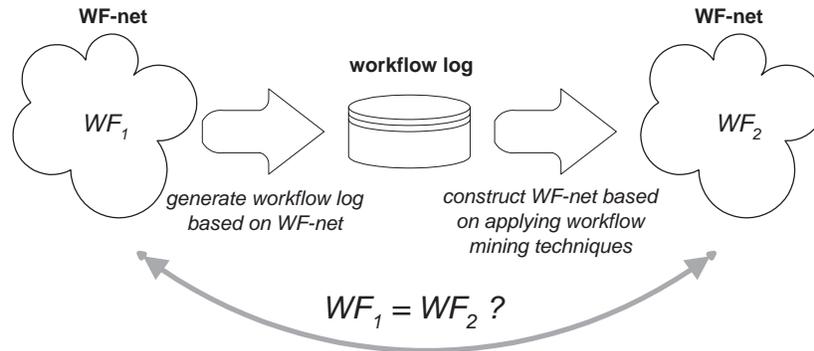

Fig. 2. The rediscovery problem: For which class of WF-nets is it guaranteed that WF_2 is equivalent to WF_1?

To illustrate the *rediscovery problem* we use Figure 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 $WF_1 = WF_2$.

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 is presented that rediscovers a large class of workflow processes. Section 5 presents a complete toolbox supporting this algorithm. 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 [10, 20, 21].

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 $P \cap T = \emptyset$, and
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 a-s, one b, and three c-s. 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., xFy). Node x is an *output node* of y iff yFx. For any $x \in P \cup T$, $\overset{N}{\bullet}x = \{y \mid yFx\}$ and $x\overset{N}{\bullet} = \{y \mid xFy\}$; the superscript N may be omitted if clear from the context.

Figure 1 shows a P/T-net consisting of 8 places and 7 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 Figure 1 (i.e., one token in the source place), transition A is enabled and firing this transition removes the token for 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 Figure 1 has 8 reachable markings. Sometimes it is convenient to know the sequence of transitions that are fired in order to reach

---

[1] In the literature, the class of Petri nets introduced in Definition 2.1 is sometimes referred to as the class of (unlabeled) *ordinary* P/T-nets to distinguish it from the class of Petri nets that allows more than one arc between a place and a transition.

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) if and only if, 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 Figure 1 is connected but not strongly connected.

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 Figure 1 is safe (and therefore also bounded) because none of the 8 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 Figure 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, 11, 16, 17]. These are used to model sequential, conditional, parallel and iterative routing (WFMC [11]). 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 post-condition 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 $o\bullet = \emptyset$,
3. *connectedness*: $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ is strongly connected,

The P/T-net shown in Figure 1 is a WF-net. Note that although the net is not strongly connected, the *short-circuited* net with transition $\bar{t}$ 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 Figure 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 towards the analysis of WF-nets is Woflan [22].

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 amongst 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.

To find a workflow model on the basis of a workflow log, the log should be analyzed for causal relations, 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 $a, b \in T$:

- $a >_W b$ if and only if 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 $t_{i+1} = b$,
- $a \rightarrow_W b$ if and only if $a >_W b$ and $b \not>_W a$,
- $a \#_W b$ if and only if $a \not>_W b$ and $b \not>_W a$, and
- $a \|_W b$ if and only if $a >_W b$ 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 *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 3.3. 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]

---

[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., $\rightarrow_W^{-1} = \{(y, x) \in T \times T \mid x \rightarrow_W y\}$.

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 3.4. ($\in$, *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*, *last* are defined as follows:

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

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 3.5. (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 if and only if $W \in \mathcal{P}(T^*)$ and every trace $\sigma \in W$ is a firing sequence of N starting in state $[i]$, i.e., $(N, [i])[\sigma\rangle$. W is a *complete* workflow log of N if and only if (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 3.6. Let $N = (P, T, F)$ be a sound WF-net and let W be a complete workflow log of N: $\{t \in T \mid \exists_{t' \in T} t >_W t' \ \lor \ t' >_W t\} = \{t \in T \mid t \notin i\bullet \cap \bullet o\}$.

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 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 $>_W$. $\qquad\square$

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

Definition 3.7. (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., $\alpha : \mathcal{P}(T^*) \rightarrow \mathcal{W}$. 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 if and only if $\alpha(W) = N$ 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 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 Figure 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 Figure 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., $\rightarrow_W$) and the presence of places connecting transitions. First, we shows that causal relations in $\rightarrow_W$ 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 $a, b \in T$: $a \rightarrow_W b$ implies $a \bullet \ \cap \ \bullet b \neq \emptyset$.

Proof. Assume $a \rightarrow_W b$ 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 $a \rightarrow_W b$. 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 the WF-net with B and C in parallel, see N_1 in Figure 4.) $W_1 = \{ABCD, ACBD\}$ is a complete log over N_1. Since $A \rightarrow_{W_1} B$, 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 Figure 4.) $W_2 = \{ABD, ACD\}$ is a complete log over N_2. Since $A \rightarrow_{W_2} B$, there has to be a place between A and B. Similarly, $A \rightarrow_{W_2} C$ and therefore there has to be a place 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 $A \rightarrow_{W_1} B$ and $A \rightarrow_{W_1} C$ correspond to two different places while in the second example the two causal relations $A \rightarrow_{W_1} B$ and $A \rightarrow_{W_1} C$ correspond to a single place.

4.2 Connecting places "often" imply causal relations

In this subsection 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 4.2. (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) if and only if, for all reachable markings $s' \in [N, s\rangle$ and transitions $t \in p\bullet$, $s' \geq \bullet t \setminus \{p\} \Rightarrow s' \geq \bullet t$.

Figure 1 contains no implicit places. However, as indicated before, adding a place p connecting transition A and B 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.
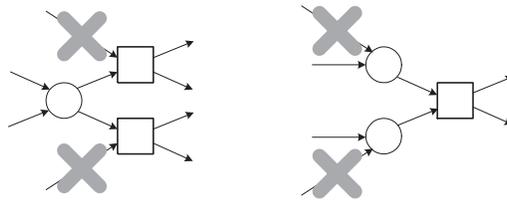

Fig. 3. Two constructs not allowed in SWF-nets.

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 Figure 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 Figure 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 4.3. (SWF-net) A WF-net $N = (P, T, F)$ is an *SWF-net* (Structured workflow net) if and only if:

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 4.3 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, 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 4.3, 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 [10]. 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 4.4. 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 4.5. 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. Let $a, b \in T$. Assume $p \in a \bullet \ \cap \ \bullet b$. We prove $a >_W b$ by considering two cases.

(i) $|p \bullet | > 1$. Consider a firing sequence σ ending with transition a. Such a firing sequence exists since N is sound. This firing sequence marks p. If p is marked, b is enabled because in an SWF-net $|p \bullet| > 1$ implies $|\bullet t| = 1$ for all transitions consuming tokens from p. Hence, $a >_W b$.

(ii) $|p \bullet | = 1$. b is the only output transition of p. If p is the only input place of b, then any occurrence of a can be followed by b and $a >_W b$. If b has multiple input places ($|\bullet b| > 1$), then the fact that N is a SWF-net implies $|\bullet p| = 1$. Therefore, a is the only transition producing tokens for p. Since p is not implicit, there is a marking $s \in [N, [i]\rangle$ such that $s \geq \bullet b \setminus \{p\}$ but not $s \geq \bullet b$, i.e., b blocks on p. Since N is sound and tokens from the input places of b can only be removed by firing b, the firing sequence leading to s can be extended to fire a directly followed by b. Hence, $a >_W b$.

$\square$

Unfortunately $a \bullet \ \cap \ \bullet b \neq \emptyset$ does not imply $a \rightarrow_W b$. To illustrate this consider Figure 4. For the first two nets (i.e., N_1 and N_2), two tasks are connected if and only if 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. Surprisingly, loops of length three or longer are no problem as is illustrated in the following theorem.

Theorem 4.6. 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. Let $a, b \in T$. Assume $a \bullet \cap \bullet b \neq \emptyset$ and $b \bullet \cap \bullet a = \emptyset$. To prove $a \rightarrow_W b$, we show that $a >_W b$ and $b \nsucc_W a$. $a >_W b$ follows directly from Theorem 4.5. Remains to prove that $b \nsucc_W a$. We will prove this by showing that it is not possible to have a firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ such that $(N, [i])[\sigma\rangle$ and $t_{n-2} = b$ and $t_{n-1} = a$. Let σ', s_n, and s_{n-2} be such that $(N, [i]) [\sigma\rangle (N, s_n)$, $\sigma' = t_1 t_2 t_3 \ldots t_{n-3}$, and $(N, [i]) [\sigma'\rangle (N, s_{n-2})$. (Note that $(N, s_{n-2}) [ba\rangle (N, s_n)$.) Let $p \in a \bullet \ \cap \ \bullet b$. In state s_{n-2}, p is marked. Moreover, a is enabled in s_{n-2} because a is enabled after firing b and $b \bullet \cap \bullet a = \emptyset$. Let s' be the marking after firing a in s_{n-2}, i.e., $(N, s_{n-2}) [a\rangle (N, s')$. If $p \notin \bullet a$, then a produces a token for p while there is a token already there, i.e., in s' place p contains at least two tokens. This is not possible since a sound WF-net is safe. Hence, there is a contradiction if $p \notin \bullet a$. If $p \in \bullet a$, then $bp \notin b \bullet$ because $b \bullet \ \cap \ \bullet a = \emptyset$. In this case, firing b disables a (i.e., $(\bullet b \setminus b \bullet) \cap \bullet a \neq \emptyset$) and thus σ is not a possible firing sequence. $\square$
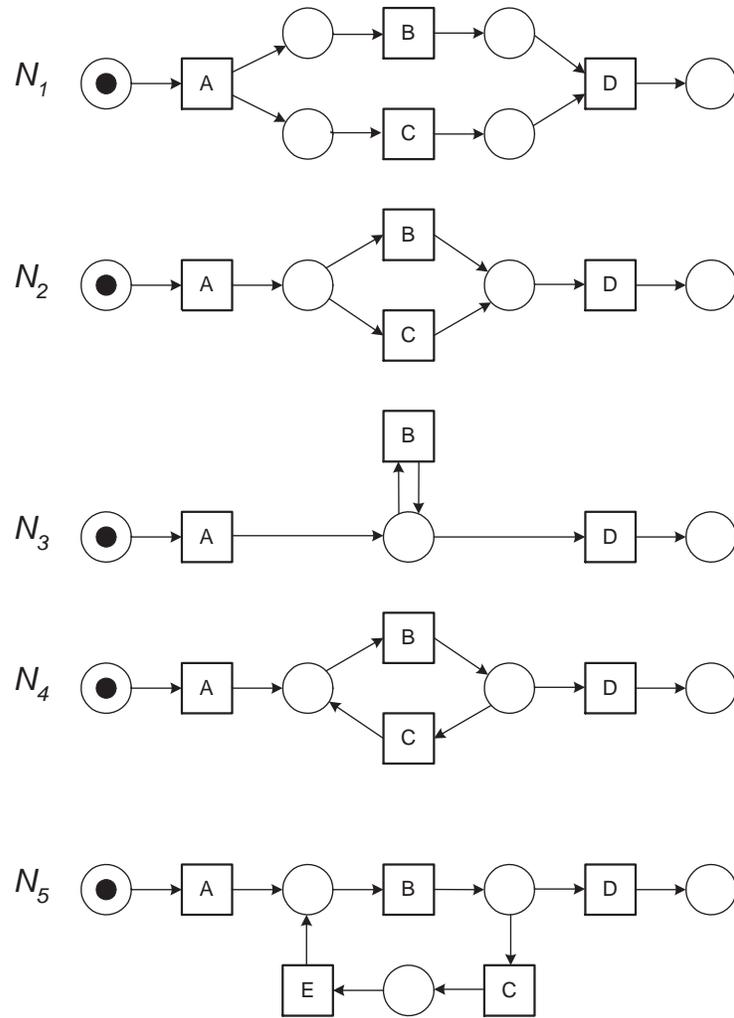
Fig. 4. Five sound SWF-nets.

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

Property 4.7. Let $N = (P, T, F)$ be an acyclic sound SWF-net and let W be a complete workflow log of N. For any $a, b \in T$: $a \bullet \ \cap \ \bullet b \neq \emptyset$ if and only if $a \rightarrow_W b$.

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 4.3. The following theorem explores the relation between the sharing of input and output places and $\#_W$.

Theorem 4.8. 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. Let $a, b, t \in T$. We prove each of the four items separately.

1. If $a \bullet \ \cap \ b \bullet \neq \emptyset$, then there is a common output place $p \in a \bullet \ \cap \ b \bullet$. If a firing of a is directly followed by b (or vice versa), then two subsequent transitions produce a token for p. These transitions do not consume tokens from p ($a \bullet \ \cap \ \bullet b = \emptyset$ or $b \bullet \ \cap \ \bullet a = \emptyset$). Therefore, p contains at least two tokens after firing a and b. This is not possible since $(N, [i])$ is safe. Hence, $a \not\rightarrow_W b$ and $b \not\rightarrow_W a$ which implies $a \#_W b$.

2. Similar arguments apply to the situation where $p \in \bullet a \ \cap \ \bullet b$.

3. Assume $a \rightarrow_W t$, $b \rightarrow_W t$, and $a \#_W b$. Theorem 4.1 implies that there are two places $p_1, p_2 \in P$ such that $p_1 \in a \bullet \cap \bullet t$ and $p_2 \in b \bullet \cap \bullet t$. Also assume that $a \bullet \cap b \bullet \cap \bullet t = \emptyset$. This implies that $p_1 \neq p_2$. We demonstrate that the latter assumption leads to a contradiction. In every complete firing sequence a, b, and t fire the same number of times because $|\bullet p_1| = |p_1 \bullet| = |\bullet p_2| = |p_2 \bullet| = 1$. In fact a and t (and b and t) fire alternatingly. Since $b \rightarrow_W t$ there is a firing sequence where a fires before b and the firing of b is directly followed by t. It is not possible that a is directly followed by b. Therefore, there is a directed path $l_{ab} \in F^*$ from a to b. If there was no directed path l_{ab}, a could be "delayed" until b becomes enabled and a firing sequence where a is directly followed by b is possible. Let L_{ab} be the set of elementary directed paths from a to b. L_{ab} is marked if one of its places contains a token and L_{ab} is unmarked if none of its places contains a token. Not every execution of a is followed by b (Since $a \rightarrow_W t$ there is a firing sequence where b fires before a and the firing of a is directly followed by t.) Therefore, there are transitions removing tokens from L_{ab} other than b. These transitions are in conflict with transitions preserving tokens for L_{ab}. However, since N is free-choice these conflicts cannot be controlled. Since these choices should be controlled depending on whether a, b or neither a nor b is the next to fire. Hence we find a contradiction.

4. Similar arguments apply to the situation where $t \rightarrow_W a$, $t \rightarrow_W b$, and $a\#_W b$.

$\square$

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 subsection.

4.3 Mining algorithm

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

Definition 4.9. (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. $Y_W = \{(A,B) \in X_W \mid \forall_{(A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \implies (A,B) = (A',B')\}$,
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 3.6, 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 if and only if $a \rightarrow_W b$. 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.

Based on α defined in Definition 4.9, we turn to the rediscovery problem. Is it possible to rediscover WF-nets using $\alpha(W)$? Consider the five SWF-nets shown in Figure 4. If 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. $\alpha(W_3)$ 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 4.10. 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 $T = T_W$. Remains to prove 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 4.9 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 4.9. 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 4.6 implies that $\forall_{a \in A}\forall_{b \in B}a \rightarrow_W b$, (2) Theorem 4.8(1) implies that $\forall_{a_1, a_2 \in A}a_1 \#_W a_2$, and (3) Theorem 4.8(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 $(A', B') \in X$ 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 $\forall_{b \in B}a' \rightarrow_W b$ and $\forall_{a \in A}a \#_W a'$. Theorem 4.8(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 4.4 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 $\forall_{a \in A}a \rightarrow_W b'$ and $\forall_{b \in B}b \#_W b'$. Using Theorem 4.8(4) and Property 4.4, 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 respectively o. This is a direct consequence of the construction given in Definition 4.9 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 $p_w\overset{\alpha(N)}{\bullet} = B$. Remains to prove

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 $a \rightarrow_W b$ and Theorem 4.1). Using Theorem 4.8, we can prove that there is just one such place. Let p be this place. Clearly, $\overset{N}{\bullet}p \subseteq A$ and $p\overset{N}{\bullet} \subseteq B$. Remains to prove 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 4.6, we can show that $a' \rightarrow_W b$. Using Theorem 4.8(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 $p\overset{N}{\bullet} = B$. $\qquad\square$
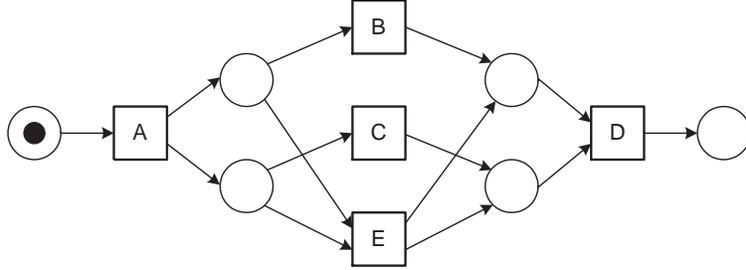

Fig. 5. Another process model corresponding to the workflow log shown in Table 1.

Nets N_1, N_2 and N_5 shown in Figure 4 satisfy the requirements stated in Theorem 4.10. Therefore, it is no surprise that α is able to rediscover these nets. The net shown in Figure 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 Figure 5. Although the net is not an SWF-net it is a sound WF-net whose observable behavior is identical to the net shown in Figure 1. Also note that the WF-net shown in Figure 5 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.

To conclude this section, we revisit the first two requirements in Definition 4.3. In Section 4.2 we already motivated the restriction to SWF-nets. To illustrate the necessity of these requirements consider figures 6 and 7. The WF-net N_6 shown in Figure 6 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 com-
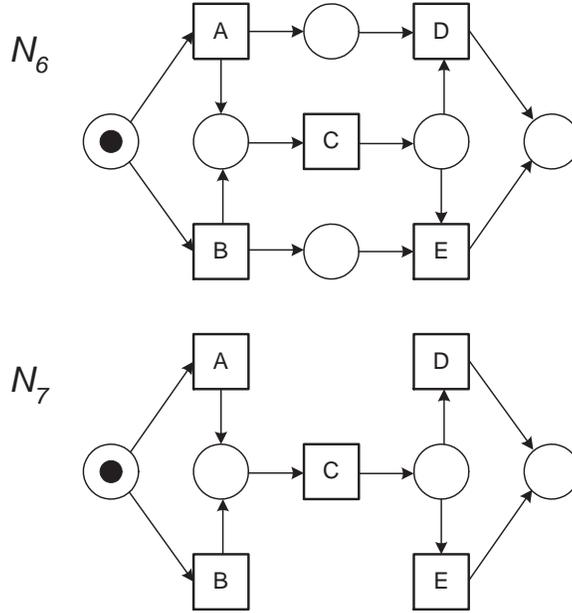
Fig. 6. The non-free-choice WF-net N_6 cannot be rediscovered.

plete workflow log W_6 of N_6, we obtain the WF-nets N_7 also shown in Figure 6 (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 4.3. The second requirement is motivated by Figure 7. N_8 violates the second requirement. If we apply the mining algorithm to a complete work-flow log W_8 of N_8, we obtain the WF-net $\alpha(W_8) = N_9$ also shown in Figure 7. Although N_9 is behaviorally equivalent, N_8 cannot be rediscovered using the mining algorithm.

Although the requirements stated in Definition 4.3 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. Even in the cases where the resulting WF-net is not be-haviorally equivalent, the results are meaningful, e.g., the process represented by N_7 is different from the process represented by N_6 (cf. Figure 6). Nevertheless, N_7 is similar and captures most of the behavior.

5 MiMo: A tool to (re)discover workflow processes

The algorithm presented in the previous section has been implemented using our tool ExSpect [3]. ExSpect (EXecutable SPECification Tool) supports high-level
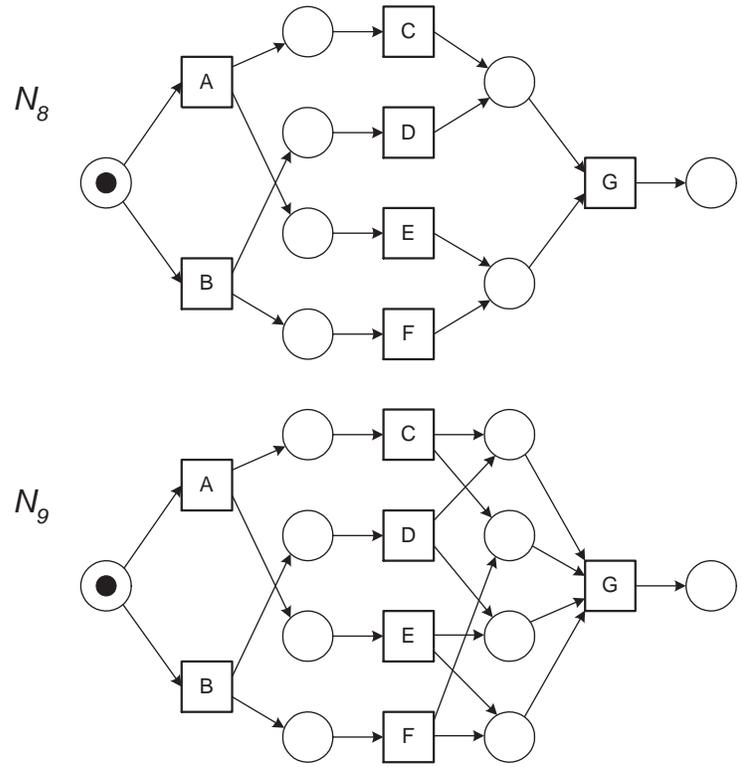
Fig. 7. WF-net N_8 cannot be rediscovered. Nevertheless α returns a WF-net which is behavioral equivalent.

Petri-nets and has been used to build a toolbox named *MiMo* (Mining Module). MiMo consists of two parts: (1) a workflow log generator and (2) a workflow log analyzer. The workflow log generator generates workflow traces on the basis of a process model. It is possible to build a graphical model of the workflow process in terms of an hierarchical WF-net. Using the MiMo toolbox a workflow log is generated automatically. The generation process can be controlled (e.g., started and stopped) by the designer. Instead of using the workflow log generator, it is also possible to upload workflow traces from a file.

The workflow log analyzer is the most interesting part of the MiMo toolbox. The analyzer is a straightforward implementation of the mining algorithm presented in the previous section. This part of the MiMo toolbox automatically generates a Petri net on the basis of a workflow log. It is possible to generate a Petri net on-the-fly and the user can inspect $\rightarrow_W$, $>_W$, $\#_W$, and $\|_W$ at any time.
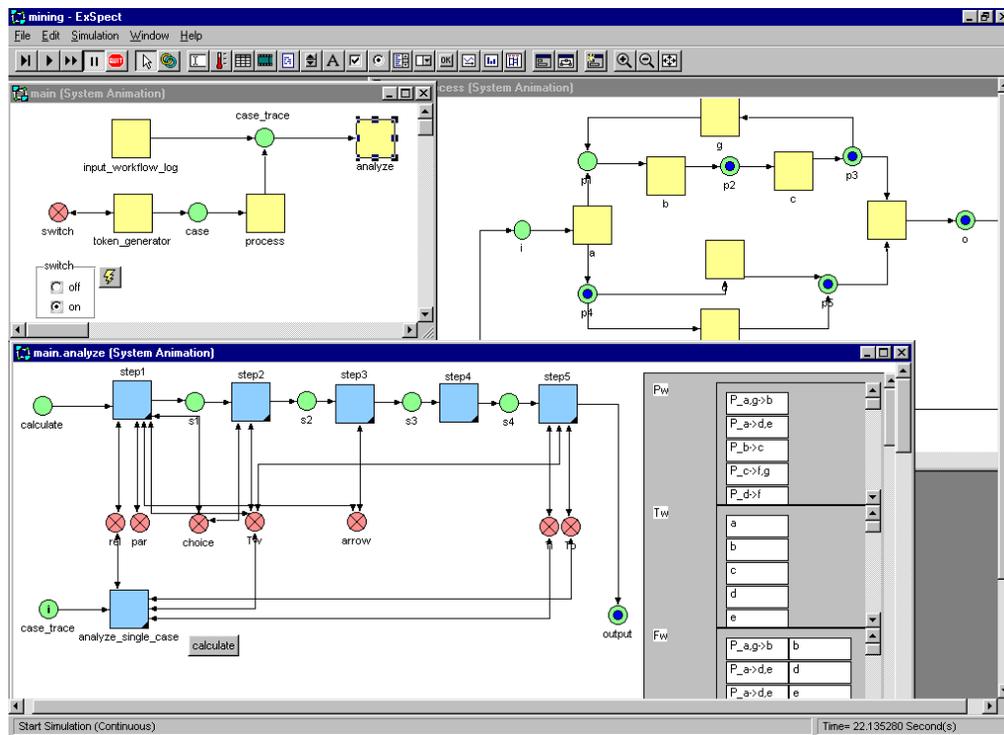

Fig. 8. A screenshot of the ExSpect module MiMo while mining a workflow process.

Figure 8 shows a screenshot of the ExSpect module MiMo. The screenshot shows the architecture (upper left window), the workflow log generator (upper

right window), and the workflow log analyzer (bottom window). All examples in this paper have been analyzed using the MiMo toolbox.

6 Related Work

The idea of process mining is not new [6–9,12–15,19]. Cook and Wolf have investigated similar issues in the context of software engineering processes. In [7] 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 [6] are limited to sequential behavior. Cook and Wolf extend their work to concurrent processes in [8]. 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 [9] 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 [6]. 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 the approach envisioned in this proposal. Given the nature of workflow graphs there is no need to identify the nature (AND or OR) of joins and splits. Moreover, workflow graphs are acyclic. The only way to deal with iteration is to enumerate all occurrences of a given activity. In [19], a tool based on these algorithms is presented. Herbst and Karagiannis also address the issue of process mining in the context of workflow management [12–15]. The approach uses the ADONIS modeling language and is based on hidden Markov models where models are merged and split in order to discover the underlying process. The work presented in [12,14,15] is limited to sequential models. A notable difference with other approaches is that the same activity can appear multiple times in the workflow model. The result in [13] incorporates concurrency but also assumes that workflow logs contain explicit causal information. The latter technique is similar to [6,19] and suffers from the drawback that the nature of splits and joins (i.e., AND or OR) is not discovered.

In contrast to existing work we addressed workflow processes with concurrent behavior right from the start (rather than adding ad-hoc mechanisms to capture parallelism), i.e., detecting concurrency is has been our prime concern in this paper. Some preliminary results have been reported in [18, 23, 24]. In [23, 24] a heuristic approach using rather simple metrics is used construct so-called "dependency/frequency tables" and "dependency/frequency graphs". In [18] another variant of this technique is presented using examples from the health-care domain. The preliminary results presented in [18, 23, 24] only provide heuristics and focus on issues such as noise. This paper differs from these approaches in the sense that we *prove* that for certain subclasses it is possible to find the right workflow model.

7 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 an algorithm that is able to rediscover a large and relevant class of workflow processes. Through examples we also showed that the algorithm provides interesting analysis results for workflow processes outside this class. In the future, we hope to improve the mining algorithm such that it is able to rediscover an even larger class of WF-nets. At this point in time, two improvements seem to be possible. First of all, it should be possible to deal with "short loops" of a particular form. Second, the rediscovery problem could be relaxed to take behaviorally equivalent WF-nets into account.

It is important to see the results presented in this paper in the context of a larger effort [18, 23, 24]. 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 [18, 23, 24]. 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 non-free-choice behavior (i.e., violating the first requirement of Definition 4.3) 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 workflow mining techniques.

At this point in time, we are applying our workflow mining techniques to two applications. The first application is in health-care where the flow of multi-disciplinary patients is analyzed. We have analyzed workflow logs (visits to different specialist) 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 multi-disciplinary patients. The second application concerns the processing of fines by the CJIB (Centraal Justitieel Incasso Bureau), the Dutch Judicial Collection Agency located in Leeuwarden. For example fines with respect to traffic viola-

tions are processed by the CJIB. However, this government agency also takes care of the collection of administrative fines related to crimes, etc. Through workflow mining we try to get insight in the life-cycle of for example speeding tickets. Some preliminary results show that it is very difficult to mine the flow of multi-disciplinary patients given the large number of exceptions, incomplete data, etc. However, it is relatively easy to mine well-structured administrative processes such as the processes within the CJIB. In both applications we are also trying to take attributes of the cases being processed into account. This way we hope to find correlations between properties of the case and the route through the workflow process.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
3. W.M.P. van der Aalst, P. de Crom, R. Goverde, K.M. van Hee, W. Hofman, H. Reijers, and R.A. van der Toorn. ExSpect 6.4: An Executable Specification Tool for Hierarchical Colored Petri Nets. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 455–464. Springer-Verlag, Berlin, 2000.
4. W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors. *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2000.
5. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
6. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
7. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
8. J.E. Cook and A.L. Wolf. Event-Based Detection of Concurrency. In *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6)*, pages 35–45, 1998.
9. J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.
10. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
11. L. Fischer, editor. *Workflow Handbook 2001, Workflow Management Coalition*. Future Strategies, Lighthouse Point, Florida, 2001.

12. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.

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

14. J. Herbst and D. Karagiannis. An Inductive Approach to the Acquisition and Adaptation of Workflow Models. In M. Ibrahim and B. Drabble, editors, *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, pages 52–57, Stockholm, Sweden, August 1999.

15. J. Herbst and D. Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 9:67–92, 2000.

16. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.

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

18. 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. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pages 183–190, 2001.

19. M.K. Maxeiner, K. Küspert, and F. Leymann. Data Mining von Workflow-Protokollen zur teilautomatisierten Konstruktion von Prozemodellen. In *Proceedings of Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 75–84. Informatik Aktuell Springer, Berlin, Germany, 2001.

20. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

21. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

22. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.

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

24. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data. In V. Hoste and G. de Pauw, editors, *Proceedings of the 11th Dutch-Belgian Conference on Machine Learning (Benelearn 2001)*, pages 93–100, 2001.