Process Mining: Extending the α-algorithm to Mine Short Loops

A.K.A. de Medeiros, B.F. van Dongen, W.M.P. van der Aalst, and A.J.M.M. Weijters

Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{a.k.medeiros, b.f.v.dongen, w.m.p.v.d.aalst, a.j.m.m.weijters}@tm.tue.nl

Abstract. The deployment of Workflow Management systems is a time-consuming and error-prone task. A possible solution is *process mining*, which automatically extracts workflow models from event-data logs. However, the current research in process mining still has problems in mining some common constructs in workflow models. Among these constructs are short loops, which are loops of length one and two. For instance, the α-algorithm was proven to mine sound Structured Workflow nets without short loops. In this paper, we present a new algorithm (the α^+-algorithm) that can handle short loops, and we prove that it correctly mines all sound Structured Workflow nets. The α^+-algorithm is based on the α-algorithm and is implemented in the EMiT tool.

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

1 Introduction

Every company wants to produce more in less time. One way to accomplish this is having a well-defined business process model that reflects the dependencies among tasks and also tasks that can be processed in parallel. *Workflow management*(WFM) systems offer the functionality to design and enact operational processes.

In an ideal situation, well-defined business processes should be designed before enactment is possible and, redesigned whenever changes happen. However, in practice a lot of time is spent on modelling business processes while the resulting workflow models are typically still error prone, because knowledge about the whole process is scattered among employees and paper procedures.

To avoid the above mentioned difficulties, instead of starting with a process design, our process mining starts by gathering information about the processes as they take place. We assume that the event data log contains enough information to correctly extract the workflow model. Any information system using transactional systems such as ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), B2B (Business to Business), SCM (Supply Chain Management) and WFM systems will offer this information in some form. Note that we do not assume that the presence of a WFM system. The only assumption we make is that it is possible to collect a process log that records the order in which the events take place.

To illustrate the principle of process mining, we consider the process log shown in Table 1. This log contains information about five cases (i.e., process

instances) and six tasks (A..F). Based on the information shown in Table 1 and by making some assumptions about the completeness of the log (i.e., if a task can follow another task, there is a log trace to show this) we can deduce for example the process model shown in Figure 1. The model is represented in terms of a Petri net [31]. After executing A, tasks B and C are in parallel. 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. Instead of starting with A the process can also start with E. Task E is always followed by task F. Table 1 contains the minimal information we assume to be present.

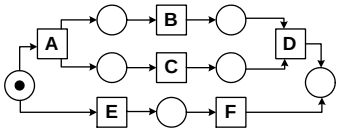

Fig. 1. A process model corresponding to the process log.

For this simple example, it is quite easy to construct a process model that is able to regenerate the process log. For larger process models this is much more difficult. For example, if the model exhibits alternative and parallel routing, then the process log will typically not contain all possible combinations. 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. These are just some of the problems that we need to face in process mining research. A lot of work is done to develop more mining algorithms that can be used in practice. In this paper a formal approach is presented. We assume perfect information: (i) the log must be complete (i.e., if a task can follow another task directly, the log contains an example of this behavior) and (ii) the log is noise free (i.e., everything that is registered in the log is correct). This is not a real limitation because we are primarily interested in formal properties of our algorithms.

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 E
case 4	task C
case 1	task D
case 3	task C
case 3	task D
case 4	task B
case 5	task F
case 4	task D

Table 1. A process log.

Process mining can be viewed as a three-phase process: *pre-processing*, *processing* and *post-processing*. In the pre-processing phase, based on the assumption that the input log contains enough information, the ordering relations be-

tween tasks are inferred. The processing phase corresponds to the execution of the mining algorithm, given the log and the ordering relations as input. In our case, the mining algorithm is the α-algorithm. During post-processing, the discovered model (in our case a Petri-net) can be fine-tuned and a graphical representation can be build.

The focus of most research in the domain of process mining is on mining heuristics based on ordering relations of the events in the process log (cf. Section 5). Considerable work has been done on heuristics to mine event-data logs to produce a process model that can support the workflow design process. However, all the existing heuristic-based mining algorithms have their limitations [26]. Typically, more advanced process constructs are difficult to handle for existing mining algorithms. Some of these problematic constructs are common in workflows and, therefore, need to be addressed to enable application in practice. Among these constructs are short loops (see Figure 2) .

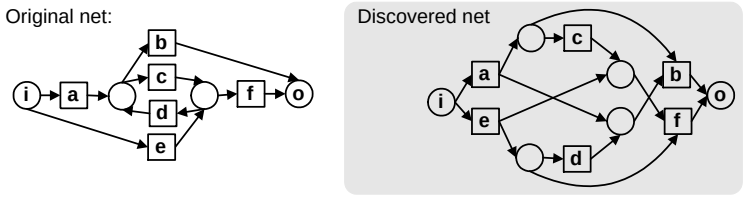

Fig. 2. Example of a sound SWF-net the α-algorithm cannot correctly mine.

The main aim of our research is to extend the class of nets we can correctly mine. The α-algorithm is proved to correctly mine sound SWF-nets without short loops [6]. In this paper we prove that it is possible to correctly mine *all nets in the class of sound SWF-nets*. The new mining algorithm is called α^+ and is based on the α-algorithm.

The remainder of this paper is organized as follows. Section 2 describes the α-algorithm and its supporting definitions. Section 3 presents the new approach to tackle length-two loops using the α-algorithm. Section 4 shows how to extend the approach in Section 3 to mine also length-one loops. Section 5 discusses how to extend the α^+-algorithm to mine nets beyond the class of sound SWF-nets. Section 6 discusses related works. Section 7 has the conclusions.

2 Preliminaries

This section contains the main definitions used in the α-algorithm that are also relevant to the α^+-algorithm. The detailed explanation about the α-algorithm and Structured Workflow Nets (SWF-nets) is in [6].

The rest of this section is as follows. Subsection 2.1 introduces standard Petri-net notations. Subsection 2.2 defines the class of WF-nets. Subsection 2.3 explains the α-algorithm.

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, 29, 31].

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 $x \in P \cup T$, ${}^{N}\!\bullet x = \{y \mid yFx\}$ and $x {\bullet\!}^{N} = \{y \mid xFy\}$; the superscript N may be omitted if clear from the context.

Figure 1 shows a P/T-net consisting of 7 places and 6 transitions. Transition A has one input place and two output places. Transition A is an *AND-split*. Transition D has two input places and one output place. Transition D is an *AND-join*. The black dot in the input place of A and E 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), transitions A and E are enabled. Although both are enabled only one can fire. If transition A fires, a token is removed from its input place and tokens are put in its output places. In the resulting marking, two transitions are enabled: B and C. Note that the firing of B and C are independent.

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

[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.

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 6 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) 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 6 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 [4, 14, 21, 23]. These are used to model sequential, conditional, parallel and iterative routing (WFMC [14]). 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, 4]. This way efficient algorithms and tools can be applied. An example of a tool tailored towards the analysis of WF-nets is Woflan [35].

Our process mining research aims at rediscovering WF-nets from event logs. However, not all places in sound WF-nets 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 2.10. (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, 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.

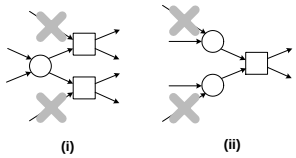

Fig. 3. Constructs not allowed in SWF-nets.

For process mining 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* and are defined as:

Definition 2.11. (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.

This paper introduces the α^+-algorithm, which mines *all* SWF-nets. The α^+-algorithm is based on the α-algorithm, which correctly mines SWF-nets *without short loops*. In our solution, we first tackle length-two loops (see Section 3) and then also length-one loops (see Section 4). While tackling length-two loops only, we do not allow the nets to have length-one loops. That is why we introduce the definition of *one-loop-free workflow nets*.

Definition 2.12. (One-loop-free workflow nets) Let $N = (P, T, F)$ be a workflow net. N is a *one-loop-free* workflow net if and only if for any $t \in T$, $t\bullet \cap \bullet t = \emptyset$.

2.3 The α-Algorithm

Our start point to mine workflows is an event log. A log is a set of traces. Workflow traces and logs are defined as:

Definition 2.13. (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]

From a worflow log, ordering relations between tasks can be inferred. In the case of the α-algorithm, every two tasks in the workflow log must have one of the following four ordering relations: $>_W$ (follows), $\rightarrow_W$ (causal), $\|_W$ (parallel) and $\#_W$ (unrelated). These ordering relations are extracted based on local information in the log traces. The ordering relations are defined as:

Definition 2.14. (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 $b >_W a$.

To ensure the workflow log contains the minimal amount of information necessary to mine the workflow, the notion of log completeness is defined as:

Definition 2.15. (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 $[i]$ and ending in state $[o]$, i.e., $(N, [i])[\sigma\rangle(N, [o])$. 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 occurs in σ.

For Figure 1, a possible complete workflow log W is: *abcd*, *acbd* and *ef*. From this complete log, the following ordering relations are inferred:

- (follows) $a >_W b$, $a >_W c$, $b >_W c$, $b >_W d$, $c >_W b$, $c >_W d$ and $e >_W f$.
- (causal) $a \rightarrow_W b$, $a \rightarrow_W c$, $b \rightarrow_W d$, $c \rightarrow_W d$ and $e \rightarrow_W f$.
- (parallel) $b \|_W c$ and $c \|_W b$.

Note that there are no *unrelated* transitions for the net in Figure 1.

Now we can give the formal definition of the α-algorithm followed by a more intuitive explanation.

Definition 2.16. (Mining algorithm α) Let W be a workflow log over T. The $\alpha(W)$ is defined as follows.

1. $T_W = \{t \in T \mid \exists_{\sigma \in W} t \in \sigma\}$,
2. $T_I = \{t \in T \mid \exists_{\sigma \in W} t = \mathit{first}(\sigma)\}$,

[2] T^* is the set of all sequences that are composed of zero of more tasks from T. $\mathcal{P}(T^*)$ is the powerset of T^*, i.e., $W \subseteq T^*$.

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 α-algorithm works as follows. First, it examines the log traces and (Step 1) creates the set of transitions (T_W) in the workflow, (Step 2) the set of output transitions (T_I) of the source place , and (Step 3) the set of the input transitions (T_O) of the sink place[3]. In steps 4 and 5, the α-algorithm creates sets (X_W and Y_W, respectively) used to define the places of the discovered workflow net. In Step 4, the α-algorithm discovers which transitions are causally related. Thus, for each tuple (A, B) in X_W, each transition in set A causally relates to *all* transitions in set B, and no transitions within A (or B) follow each other in some firing sequence. These constraints to the elements in sets A and B allow the correct mining of AND-split/join and OR-split/join constructs. Note that the OR-split/join requires the fusion of places. In Step 5, the α-algorithm refines set X_W by taking only the largest elements with respect to set inclusion. In fact, Step 5 establishes the exact amount of places the discovered net has (excluding the source place i_W and the sink place o_W). The places are created in Step 6 and connected to their respective input/output transitions in Step 7. The discovered workflow net is returned in Step 8.

Finally, we define what it means for a WF-net to be rediscovered.

Definition 2.17. (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.

3 Length-Two Loops

In this section we first show why a new notion of log completeness is necessary to capture length-two loops in SWF-nets and why the α-algorithm does not capture length-two loops in SWF-nets (even if the new notion of log completeness is used). Then a new definition of ordering relations is given, and finally we prove that this new definition of ordering relations is sufficient to tackle length-two loops with the α-algorithm.

[3] In a workflow net, the source place i has no input transitions and the sink place o has no output transitions.

3.1 New Notion of Log Completeness

Log completeness as defined in Definition 2.15 is insufficient to detect length-two loops in SWF-nets. As an example, consider the SWF-net in Figure 2 (left-hand side). This net can have the complete log: *ab, acdb, edcf, ef*. However, by looking at this log it is not clear if transitions c and d are in parallel or belong to a length-two loop. Thus, to correctly detect length-two loops in SWF-nets, the following new definition of complete log is introduced.

Definition 3.1. (Loop-complete workflow log) Let $N = (P, T, F)$ be a sound SWF-net and W a log of N. W is a *loop-complete workflow log of* N if and only if W is complete and for all workflow logs W' of N: if there is a firing sequence $\sigma' \in W'$ with $\sigma' = t_1 t_2 t_3 \dots t_{n'}$ and $i' \in \{1, \dots, n' - 2\}$ such that $t_{i'} = t_{i'+2} = a$ and $t_{i'+1} = b$, for some $a, b \in T : a \neq b$, then there is a firing sequence $\sigma \in W$ with $\sigma = t_1 t_2 t_3 \dots t_n$ and $i \in \{1, \dots, n - 2\}$ such that $t_i = t_{i+2} = a$ and $t_{i+1} = b$.

Note that a *loop-complete workflow log* for the net in Figure 2 will contain one or more traces with the substrings "*cdc*" and "*dcd*". Besides, all loop-complete workflow logs are also complete workflow logs.

In definition 3.1 it is implicitly assumed that there exists a loop-complete workflow log with a finite set of traces. Furthermore, it is assumed that all traces are finite. We would like to point out that it is indeed possible to have a loop-complete workflow log with these properties.

Theorem 3.2. It is possible to have a loop-complete log that is a finite set of finite traces.

Proof. Consider the reachability graph of a SWF-net. Since the number of possible markings is finite, the reachability graph is also finite. Furthermore, soundness (or more specifically liveness) implies that in the reachability graph there is a path from marking $[i]$ to every other reachable marking and there is a path from every reachable marking to the state $[o]$. Every arc in the reachability graph can be mapped onto the firing of a transition, therefore we need to show two things:

- For every two arcs (s_{j-1}, s_j) and (s_j, s_{j+1}) there is a path from $[i]$ to $[o]$ over these arcs. Liveness properties of a sound SWF-net give that there is a shortest path from marking $[i]$ to marking s_{j-1}. Furthermore, there is a shortest path from marking s_{j+1} to marking $[o]$. Combining these two paths with the two arcs (s_{j-1}, s_j) and (s_j, s_{j+1}) leads to a finite path from $[i]$ to $[o]$ over these two arcs. Extending this to all pairs shows that there are finite traces that contain all information for the $>$ relation.
- For every three arcs (s_{j-1}, s_j), (s_j, s_{j+1}) and (s_{j+1}, s_{j+2}) such that (s_{j-1}, s_j) and (s_{j+1}, s_{j+2}) represent the firing of the same transition t_1 and (s_j, s_{j+1}) represents the firing of a different transition t_2, we need to show that there is again a path from $[i]$ to $[o]$ over these arcs. The proof for this is the same as for the $>$ relation. Therefore, there also exist finite traces containing all the information needed to detect the firing sequences $t_1 t_2 t_1$.

All that we need to show now is that it is possible to generate a finite set of finite traces that is a loop-complete log. Since we have shown that for each pair or triple of arcs it is possible to generate a finite trace, and the number of pairs and triplicates is bounded in the size of the graph, we know that this is the case. □

3.2 Redefinition of Ordering Relations

The new notion of a loop-complete workflow log is necessary but not sufficient to mine length-two loops. The main reason is that the tasks in the length-two loop are inferred to be in parallel. For example, for the net in Figure 2, any loop-complete workflow log will lead to $c\|_W d$ and $d\|_W c$. However, these transitions are not in parallel. In fact, they are connected by places that can only be correctly mined by the α-algorithm if at least $c \rightarrow_W d$ and $d \rightarrow_W c$. Using this insight, we redefine Definition 2.14, i.e., we provide the following new definitions for the basic ordering relations $\rightarrow_W$ and $\|_W$.

Definition 3.3. (Ordering relations capturing length-two loops) Let W be a loop-complete workflow log over T, i.e., $W \in \mathcal{P}(T^*)$. Let $a, b \in T$:

- $a \triangle_W b$ if and only if there is a trace $\sigma = t_1 t_2 t_3 \dots t_n$ and $i \in \{1, \dots, n-2\}$ such that $\sigma \in W$ and $t_i = t_{i+2} = a$ and $t_{i+1} = b$,
- $a \diamond_W b$ if and only if $a \triangle_W b$ and $b \triangle_W a$,
- $a >_W b$ if and only if there is a trace $\sigma = t_1 t_2 t_3 \dots t_{n-1}$ and $i \in \{1, \dots, n-2\}$ 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 ($b \not>_W a$ or $a \diamond_W b$) ,
- $a \#_W b$ 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 $b >_W a$ and $a \not\diamond_W b$.

Note that, in the new Definition 3.3, two transitions a and b are also in the $a \rightarrow_W b$ relation if $a >_W b$ and $b >_W a$ and the substrings aba and bab are contained in the log traces.

Theorem 3.4. Let $N = (P, T, F)$ be an one-loop-free sound SWF-net. Let W be a loop-complete workflow log of N. For any $a, b \in T$, such that $\bullet a \cap b \bullet \neq \emptyset$ and $a \bullet \cap \bullet b \neq \emptyset$, $a \triangle_W b$ implies $b \triangle_W a$.

Proof. Assume $a \bullet \cap \bullet b \neq \emptyset$, $b \bullet \cap \bullet a \neq \emptyset$, $\exists \sigma = ...aba...$ and $\nexists \sigma' = ...bab....$ We show that this leads to a contradiction.

Since N is a SWF-net and no implicit places are possible, $|a \bullet \cap \bullet b| = 1$ and $|b \bullet \cap \bullet a| = 1$. Let $p_{ab}, p_{ba} \in P$ be the places between a and b, such that $a \bullet \cap \bullet b = \{p_{ab}\}$ and $b \bullet \cap \bullet a = \{p_{ba}\}$. Since $\exists \sigma = ...aba...$ and N is safe, $|\bullet a| = 1$ and $\bullet a = \{p_{ba}\}$. Since there is no firing sequence $\sigma' = ...bab...$ and $a \bullet \cap \bullet b \neq \emptyset$, b has more than one input place, i.e. $|\bullet b| > 1$. We define $\{p_{ab}, p_{b,in}\} \subseteq \bullet b$.

Consequently, the following properties hold:

- $|\bullet p_{ab}| = 1$. This is a direct consequence of Definition 2.11(2).
- $|p_{ab} \bullet| = 1$. This is a direct consequence of Definition 2.11(1).

– $p_{b,in}\bullet = \{b\}$. This is a direct consequence of Definition 2.11(1).

By definition, $\bullet p_{ab} = \{a\}$ and $p_{ab}\bullet = \{b\}$. Since the net is safe and the sequence $\sigma = ...aba...$ exists, we know that all tokens produced by a must be consumed by b. From the definition of SWF-nets it is known that $|a\bullet \cap \bullet b| \leq 1$ for all a and b. Therefore, $a\bullet = \{p_{ab}\}$ meaning that the firing of a can only affect the enabling of b.
Furthermore, it can be shown that $\{p_{ba}\} \subset b\bullet$. Assume that $\{p_{ba}\} = b\bullet$. We show that this leads to a contradiction. Since the net is live, there is a marking s_1 that puts a token in p_{ab} and $p_{b,in}$ to enable b. Since the token in p_{ab} can only arrive after firing a, a marking s_2 that puts a token in p_{ba} and $p_{b,in}$ must also exist. Assume that there is a trace containing b, i.e. $\exists\sigma_1 = ...ab\sigma_1'$ such that $b \notin \sigma_1' \wedge a \notin \sigma_1'$. From the property of *proper completion* we know that such a trace must exist. We know that no transition can remove tokens from $p_{b,in}$ other than b. Since the trace σ_1' can be fired from a marking s marking p_{ba} and not $p_{b,in}$, it can be fired from a marking s' marking every place that is marked in s and marking $p_{b,in}$. However, after firing σ_1' from marking s' a token is eventually placed in the sink place and the token in $p_{b,in}$ is not removed. This is a contradiction to the property of *proper completion* of SWF-nets.
Now we know that there must be a place $p_{b,out}$ such that $\{p_{ba}, p_{b,out}\} \subseteq b\bullet$ and $p_{b,out} \neq p_{b,in}$. From the property of *safeness* we know that after firing b the token from $p_{b,out}$ has to be removed before b becomes enabled again. This means that from the moment b becomes enabled there are two tokens travelling. One between p_{ab} and p_{ba} and one between $p_{b,in}$ and $p_{b,out}$. Since the property of *proper completion* holds for the net, we know that these two tokens need to be synchronized in order to exit the loop between a and b.

Structural properties of SWF-nets forbid a transition to take tokens from p_{ab} and tokens in $p_{b,in}$ can only be consumed by b. *Proper completion* implies that it has to be possible to remove a token from p_{ba}. Any transition removing this token does not have any other input places because of structural properties. *Liveness* implies that the token produced in $p_{b,out}$ can be transported back to $p_{b,in}$ by firing at least one transition. The subnet transporting this token cannot know whether transition a has fired, or the token in p_{ba} has been removed never to return. Any construction that checks this condition would namely violate free-choiceness. Therefore, this subnet could produce a token in $p_{b,in}$, even though transition a will never become enabled again. This violates *proper completion* or *deadlock freedom* and therefore, a net where $p_{ab} \subset \bullet b$ cannot be a sound SWF-net. Therefore, $\bullet b = \{p_{ab}\}$ and consequently, $b\bullet = \{p_{ba}\}$. However, if that is the case then the trace $\sigma' = ...bab...$ is possible, thus leading to a contradiction.

□

However, there is still a problem. Length-one loops in the net may also produce "*cdc*" and "*dcd*" patterns in the log traces. For example, see the nets in Figure 4. Therefore, to prove that the α-algorithm can correctly mine length-two loops when using the new definitions of loop-complete workflow log and ordering relations, we assume that the net is a *sound one-loop-free SWF-net*. In Section 4, we show how to handle the general case (all sound SWF-nets).

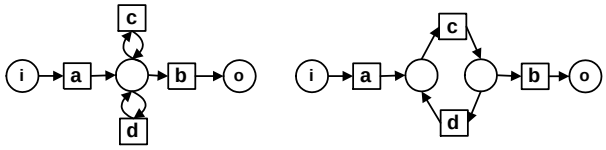

Fig. 4. Example illustrating why length-one loops are not allowed when mining length-two loops. Note that both nets have loop-complete workflow logs that contains traces with the substrings "cdc" or "dcd".

3.3 Proof that the α-algorithm Mines Length-Two Loops

In [6], the α-algorithm was proved to correctly mine sound SWF-nets without short loops. In subsections 3.1 and 3.2, we respectively introduced the new definitions of log completeness (Definition 3.1) and ordering relations (Definition 3.3) that are necessary to capture length-two loops using the α-algorithm. However, we need to prove that the α-algorithm correctly mines one-loop-free sound SWF-nets when the definitions 3.1 and 3.3 are used. Thus, this subsection has proofs for the relevant theorems and properties in [6] that do not hold directly when using the definitions 3.1 and 3.3.

Theorem 3.5. [4] Let $N = (P, T, F)$ be a sound one-loop-free SWF-net and let W be a loop-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 $a \rightarrow_W b$. If $a \not> _W b$ then $a >_W b$ and $b >_W a$ and Theorem 4.1 in [6] can be used to prove that $a\bullet \cap \bullet b \neq \emptyset$. Therefore, we also assume that $a \diamond_W b$. Now we need to find a contradiction when we also assume that $a\bullet \cap \bullet b \neq \emptyset$. Because N is one-loop-free and safe, for any $t \in T$, $t\bullet \cap \bullet t = \emptyset$ and there is no firing sequence $\sigma'' = ...tt...$. So, once a transition t fires, it can be enabled again only after the firing of another distinct transition. However, this leads to a contradiction because there is a firing sequence $\sigma' = ...bab...$ that cannot be true if $a\bullet \cap \bullet b = \emptyset$, since a is the only transition to fire between the two firings of b. For similar reasons, it is also impossible to have $\sigma = ...aba...$ if $\bullet a \cap b\bullet = \emptyset$. □

Theorem 3.6. [5] Let $N = (P, T, F)$ be a sound one-loop-free SWF-net and let W be a loop-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.

[4] Theorem 4.1 in [6].
[5] Theorem 4.8 in [6].

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 (because $a\bullet \cap \bullet a = \emptyset$ and $b\bullet \cap \bullet b = \emptyset$). Therefore, p contains at least two tokens after firing a and b. This is not possible since $(N, [i])$ is safe. Hence, $a \not>_W b$ and $b \not>_W a$ which implies $a \#_W b$.
2. Similar arguments apply to the situation where $p \in \bullet a \cap \bullet b$.
3. Assume $a \rightarrow_W t$, $b \rightarrow_W t$, $a \#_W b$ and $a\bullet \cap b\bullet \cap \bullet t = \emptyset$. We show this leads to a contradiction. From Theorem 3.5, we know $a \rightarrow_W t$ implies $a\bullet \cap \bullet t \neq \emptyset$. Similarly, $b \rightarrow_W t$ implies $b\bullet \cap \bullet t \neq \emptyset$. We assumed that $a\bullet \cap b\bullet \cap \bullet t = \emptyset$. Thus $a\bullet \cap b\bullet = \emptyset$. Moreover, if $p_{at} \in a\bullet \cap \bullet t$ and $p_{bt} \in b\bullet \cap \bullet t$, then $\bullet p_{at} \cap \bullet p_{bt} = \emptyset$. This means that t can only fire after the firings of *both* a and b. If $a \triangle_W t$, we know there is a firing sequence $\sigma' = ...tat....$ This means that t fired (afterwards all its input places are empty since the net is safe), a fired and t fired again. However, this is impossible since $b\bullet \cap \bullet t \neq \emptyset$ and $a\bullet \cap b\bullet = \emptyset$, i.e. p_{bt} is empty. So $a\bullet \cap b\bullet \cap \bullet t \neq \emptyset$ if $a \triangle_W t$. Similar arguments hold if $b \triangle_W t$, $t \triangle_W a$, and $t \triangle_W b$. Therefore, we can refer to the proof at Theorem 4.8 in [6].
4. Similar arguments apply to the situation where $t \rightarrow_W a$, $t \rightarrow_W b$, and $a \#_W b$.

□

Theorem 3.7. Let $N = (P, T, F)$ be a sound one-loop-free SWF-net and let W be a loop-complete workflow log of N. Then $\alpha(W) = N$ modulo renaming of places.

Proof. This theorem is correct because all theorems and properties of the original α-algorithm are proven to still hold when dealing with sound one-loop-free SWF-net, loop-complete workflow logs and the new ordering relations. □

4 Length-One Loops

In this section we show the properties of length-one loops in sound SWF-nets and an algorithm (called α^+) that correctly mines all sound SWF-nets.

4.1 Properties of Length-One Loops

Length-one loops are connected to a single place in any sound SWF-net and this place cannot be the source or sink place of the sound SWF-net, as is stated in the following theorem:

Theorem 4.1. Let $N = (P, T, F)$ be a sound SWF-net. For any $a \in T$, $a\bullet \cap \bullet a \neq \emptyset$ implies $a \notin i\bullet$, $a \notin \bullet o$, $a\bullet = \bullet a$ and $|\bullet a| = 1$.

Proof. Let $a \in T$. We prove by contradiction. Assume $a\bullet \cap \bullet a \neq \emptyset$ and:

1. $a \in i\bullet$. If $|\bullet a| = 1$, then $\bullet a = a\bullet = i$, since $a\bullet \cap \bullet a \neq \emptyset$. However, N is a WF-net and $\bullet i = \emptyset$ (see Definition 2.8(1)). So we have a contradiction.

2. $a \in \bullet o$. Again if $|\bullet a| = 1$ we have a contradiction (see Definition 2.8(2)).
3. $\bullet a \neq a\bullet$, i.e. there is a place $p \in \bullet a \setminus a\bullet$ or $p \in a\bullet \setminus \bullet a$. If $p \in \bullet a \setminus a\bullet$, then a is dead because the SWF-net properties imply that the places in $a\bullet \cap \bullet a$ can never be marked. If $p \in a\bullet \setminus \bullet a$, the resulting net is unbounded because if a can fire once it can fire multiple times.
4. $|\bullet a| > 1$. Let $p \in a\bullet = \bullet a$. Again, from Definition 2.11(2), $|\bullet a| > 1$ implies $|\bullet p| = 1$. But if this is the case, a is a deadlock because the only transition that can put a token in p is a. This is a contradiction since N is sound.

□

Property 4.2. Let $N = (P, T, F)$ be a sound SWF-net. Let W be a loop-complete workflow log of N. For any $a \in T : \bullet a \cap a\bullet \neq \emptyset$ implies there are $b, c \in T : a \neq b$ and $b \neq c$ and $a \neq c$ and $b \rightarrow_W a$ and $a \rightarrow_W c$ and $b \rightarrow_W c$ and $\bullet c = \bullet a$.

This property follows directly from Definition 2.11(1) and Theorem 4.1 in this paper, and from Theorem 4.6[6] in [6]. Property 4.2 states there are always non-length-one loops transitions that are either input transitions or output transitions from the single place connected to a length-one-loop transition in a sound SWF-net. Furthermore, the output transitions have only this single place as their input place.

Theorem 4.3. Let $N = (P, T, F)$ be a sound SWF-net. Let $N' = (P', T', F')$ be a one-loop-free PT-net such that $P' = P$, $T' = \{t \in T | \bullet t \cap t\bullet = \emptyset\}$, and $F' = F \cap (P' \times T' \cup T' \times P')$. Let W be a loop-complete workflow log of N and let W^{-L1L} be the log created by excluding the occurrences of length-one-loop transitions from every log trace in W. Then:

1. N' is a sound one-loop-free SWF-net,
2. $\alpha(W^{-L1L}) = N'$ modulo renaming of places.

Proof. We prove (1) by checking if N' is sound, one-loop-free and a SWF-net. We prove (2) by showing W^{-L1L} is a loop-complete workflow log of N'.

1. N' is a one-loop-free by definition. Thus, it remains to prove N' is sound and is a SWF-net. Let us first look at the soundness of N'. From Theorem 4.1, we know any length-one-loop transition t is connected to a single place in a sound SWF-net. Thus, the firing of t does not change the marking of N. We know N' is originated from N by excluding all t and the arcs connected to t. So the set of reachable markings for N coincides with the set of reachable markings for N'. Consequently, N' is sound because N is sound. By reasoning about the set of reachable markings is also easy to see N' is a WF-net. It remains to check if N' is a SWF-net. Definition 2.11(1) and 2.11(2) hold directly because N' is created by only excluding transitions and arcs from N, which is already a SWF-net. Definition 2.11(1) holds because,

[6] This theorem is defined as:
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 $a \rightarrow_W b$.

as stated in Property 4.2, for any other transition t' such that $\bullet t' \cap t'\bullet = \emptyset$ and $\bullet t \cap \bullet' t \neq \emptyset$ implies t' has a single input place p that is also connected to t. Note that t' exists because $p \neq o$. Since $|\bullet t| = 1$, p can never become an implicit place in N'.

2. Let W' be a loop-complete workflow log of N'. N' and N have the same set of reachable markings. Thus, for any two transitions $a, b \in T'$, if $a >_{W'} b$ holds, then $a >_W b$ also holds. Since W^{-L1L} is created by excluding the occurrences of length-one-loop transitions t (i.e. $\bullet t \cap t\bullet \neq \emptyset$) from the log traces of W, $a >_{W^{-L1L}} b$ also holds. Consequently, W^{-L1L} is a loop-complete workflow for N'. Since N' was proven in (1) to be a sound one-loop-free SWF-net, by Theorem 3.7, $\alpha(W^{-L1L}) = N'$.

□

Theorem 4.3 states that the main net structure (called N' in the theorem) of any sound SWF-net can be correctly discovered by the α-algorithm whenever length-one-loop transitions are removed from the input log. Consequently, since length-one-loop transitions are always connected to a single place in sound SWF-net (Theorem 4.1), we can use the α-algorithm to mine the main net structure N' and then connect the length-one-loop transition to this net.

The next subsection shows how to identify length-one-loops transitions and how to correctly insert them in the net the α-algorithm outputs.

4.2 Solution to Tackle Length-One Loops

The solution to tackle length-one loops in sound SWF-nets focusses on the pre- and post-processing phases of process mining. The key idea is to identify the length-one-loop tasks and the single place to which each task should be connected. Any length-one-loop task t can be identified by searching a loop-complete workflow log for traces containing the substring tt. To determine the correct place p to which each t should be connected in the discovered net, we must check which transitions are directed followed by t but do not direct follow t (i.e. p is an output place of these transitions) and which transitions direct follow t but t does not direct follow them (i.e. p is the input place of these transitions).

The algorithm - called α^+ - to mine sound SWF-nets is formalized as follows. Note that the function *eliminateTask* maps any log trace σ to a new one σ' without the occurrence of a certain transition t.

Definition 4.4. (Mining algorithm α^+) Let W be a loop-complete workflow log over T, the α-algorithm as in Definition 2.16 and the ordering relations as in Definition 3.3.

1. $T_{log} = \{t \in T | \ \exists_{\sigma \in W}[t \in \sigma]\}$
2. $L1L = \{t \in T_{log} \mid \exists_{\sigma = t_1 t_2 \ldots t_n \in W; i \in \{1,2,\ldots,n\}}[t = t_{i-1} \wedge t = t_i]\}$
3. $T' = T_{log} \setminus L1L$
4. $F_{L1L} = \emptyset$
5. For each $t \in L1L$ do:

(a) $A = \{a \in T' \mid a >_W t\}$

(b) $B = \{b \in T' \mid t >_W a\}$

(c) $F_{L1L} := F_{L1L} \cup \{(t, p_{(A \setminus B, B \setminus A)}), (p_{(A \setminus B, B \setminus A)}, t)\}$

6. $W^{-L1L} = \emptyset$
7. For each $\sigma \in W$ do:

   (a) $\sigma' = \sigma$

   (b) For each $t \in L1L$ do:

      i. $\sigma' := eliminateTask(\sigma', t)$

   (c) $W^{-L1L} := W^{-L1L} \cup \sigma'$
8. $(P_{W^{-L1L}}, T_{W^{-L1L}}, F_{W^{-L1L}}) = \alpha(W^{-L1L})$
9. $P_W = P_{W^{-L1L}}$
10. $T_W = T_{W^{-L1L}} \cup L1L$
11. $F_W = F_{W^{-L1L}} \cup F_{L1L}$
12. $\alpha^+ = (P_W, T_W, F_W)$

The α^+ works as follows. First, it examines the log traces (Step 1) and identifies the length-one-loop transitions (Step 2). In steps 3 to 5, the places to which each length-one-loop transition should be connected to are identified and the respective arcs are included in F_{L1L}. Then, all length-one-loop transitions are removed from the input log W^{-L1L} to be processed by the α-algorithm (steps 6 and 7). In Step 8, the α-algorithm discovers a workflow net based on the loop-complete workflow log W^{-L1L} and the ordering relations as defined in Definition 3.3. In steps 9 to 11, the length-one-loop transitions and their respective input and output arcs are added to the net discovered by the α-algorithm. The workflow net with the added length-one loops is returned in Step 12.

Theorem 4.5. Let $N = (P, T, F)$ be a sound SWF-net and let W be a loop-complete workflow log of N . Using the ordering relations as in Definition 3.3, $\alpha^+(W) = N$ modulo renaming of places.

Proof. If N is an one-loop-free sound SWF-net, the α^+-algorithm works as the α-algorithm. If N is not an one-loop-free sound SWF-net, given Theorem 4.3, it suffices to prove F_{L1L} correctly connects all length-one-loops transitions to the discovered net $(P_{W'}, T_{W'}, F_{W'})$. We prove this in two parts: first we prove the arcs $(t, p_{(A \setminus B, B \setminus A)})$ and $(p_{(A \setminus B, B \setminus A)}, t)$, created in Step 5c, point to places in $P_{W^{-L1L}}$. In other words, $p_{(A \setminus B, B \setminus A)} \in P_{W^{-L1L}}$. Second we prove $p_{(A \setminus B, B \setminus A)}$ is the right place.

Part 1: Arcs are connected to existing places. We prove this by showing $(A \setminus B, B \setminus A) \in X_W$ (Definition 2.16(4)) in α-algorithm. Let $t \in L1L$ be a length-one-loop transition. From Property 4.2, we know there are transitions $x, y \in T'$, such that $x \rightarrow_W t$, $t \rightarrow_W y$, $x \rightarrow_W y$ and $\bullet y = \bullet t$. $x \in A \setminus B$ because $x >_W t$ and $t \not>_W x$. Similarly, $y \in B \setminus A$. Because $x \rightarrow_W y$, $x \in A \setminus B$ and $y \in B \setminus A$, we can conclude $(A \setminus B) \rightarrow_W (B \setminus A)$. To show $(A \setminus B, B \setminus A) \in X_W$, it remains to prove that for any $x_1, x_2 \in A \setminus B$, $x_1 \#_W x_2$ and for any $y_1, y_2 \in B \setminus A$, $y_1 \#_W y_2$. But

this follows directly from Theorem 3.6(1-2). Note that $x_1\bullet \cap x_2\bullet \neq \emptyset$ because $x_1\bullet \cap \bullet t \neq \emptyset$, $x_2\bullet \cap \bullet t \neq \emptyset$ and $|\bullet t| = 1$. Similar reasoning applies to y_1 and y_2.

Part 2: Here we need to prove that the arcs are connected to the right place. This means $(A \setminus B, B \setminus A) \in Y_W$ (Definition 2.16(5)) in the α-algorithm. Let us first check if $B \setminus A$ is maximal. Let $t \in L1L$ be a length-one-loop transition. Assume there exists $y \in T'$ such that $(A \setminus B) >_W y$, $y \in B$, $\bullet t \cap \bullet y \neq \emptyset$ and $y \notin B \setminus A$. Since $y \in B$ but $y \notin B \setminus A$, then $y \in A$ and $y >_W t$. However, this is a contradiction because $y\bullet \cap \bullet\, y \neq \emptyset$ (from $y \in T'$) and $\bullet t \cap \bullet y \neq \emptyset$ imply $y\bullet \cap \bullet t \neq \emptyset$. Thus, when y fires it consumes the token in $\bullet t$ and does not return it. Since we are dealing with logs of safe nets, $y >_W t$ is impossible. A similar reasoning is used to prove $A \setminus B$ is maximal. Since both $A \setminus B$ and $B \setminus A$ are proven to be maximal, $A \setminus B, B \setminus A) \in Y_W$ and the arcs in F_{L1L} are connected to the right place. □

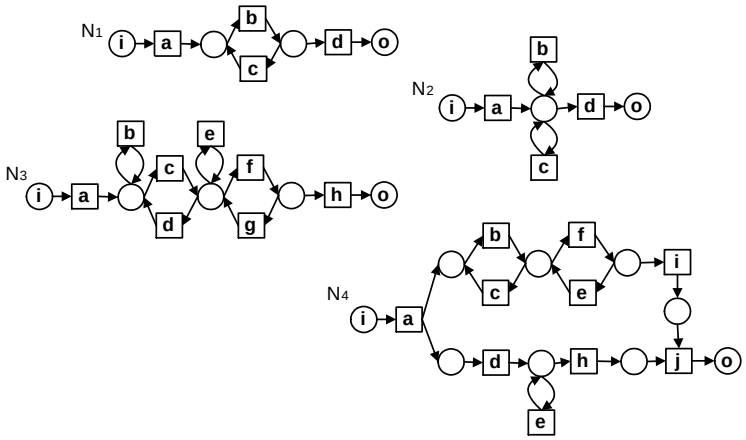

Fig. 5. Examples of sound SWF-nets that the α^+-algorithm correctly mines.

The original net in Figure 2 and the nets N_{1-4} in Figure 5 satisfy the requirements stated in Theorem 4.5. Therefore, they are all correctly discovered by the α^+-algorithm. In fact, the α^+ can be extended to correctly discover nets beyond the class of sound SWF-net. These nets are discussed in the next section.

5 Extension beyond SWF-nets

To enable the α^+-algorithm to correctly discover nets beyond the class of sound SWF-net, Step 11 in Definition 4.4 must be modified to:

11. $F_W = F_{W'} \cup$
$\{(t, p_{(A,B)}) \in (L1L \times P_W) \mid \exists_{(t', p_{(A',B')}) \in F_{L1L}}[t = t' \ \wedge \ A \subseteq A' \ \wedge \ B \subseteq B']\} \cup$
$\{(p_{(A,B)}, t) \in (P_W \times L1L) \mid \exists_{(p_{(A',B')}, t') \in F_{L1L}}[t = t' \ \wedge \ A \subseteq A' \ \wedge \ B \subseteq B']\}$

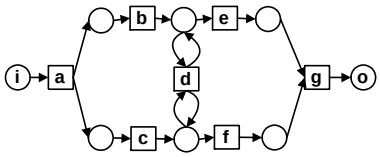

Fig. 6. Example of a sound WF-net that the *modified version* of the α^+-algorithm (the α^{++}) correctly mines.

As an example, see the net in Figure 6. This net is not a SWF-net, but it is correctly mined by the modified version of the α^+-algorithm. Let us call α^{++} the modified version of the α^+-algorithm. Note that the α^{++}-algorithm behaves exactly as the α^+-algorithm when dealing with loop-complete workflow logs of sound SWF-nets. This happens because Step 11 in Definition 4.4 can be rewritten to:

$F_W = F_{W'} \cup$
$\{(t, p_{(A,B)}) \in (L1L \times P_W) \mid \exists_{(t', p_{(A',B')}) \in F_{L1L}} [t = t' \ \wedge \ A = A' \ \wedge \ B = B']\} \cup$
$\{(p_{(A,B)}, t) \in (P_W \times L1L) \mid \exists_{(p_{(A',B')}, t') \in F_{L1L}} [t = t' \ \wedge \ A = A' \ \wedge \ B = B']\}$

The argument above makes it trivial to see that all the proofs given in Section 4 are valid when the α^{++}-algorithm is used. In fact, the EMiT mining tool implements the α^{++}-algorithm.

We think the α^{++}-algorithm correctly mines all *sound* nets of the following class:

Definition 5.1. (Extended SWF-nets) A WF-net $N = (P, T, F)$ is an *extended SWF-net* (Structured workflow net) if, and only if:

1. For all $p \in P$ and $t \in T$ with $(p, t) \in F$ and $\bullet t \cap t\bullet = \emptyset$: $|p \bullet| > 1$ implies $|\bullet t| = 1$.
2. For all $p \in P$ and $t \in T$ with $(p, t) \in F$ and $\bullet t \cap t\bullet = \emptyset$: $|\bullet t| > 1$ implies $|\bullet p| = 1$.
3. There are no implicit places.
4. For all $t \in T : \bullet t \cap t\bullet \neq \emptyset$ implies $\bullet t = t\bullet$ and (for all $t' \in T : t' \bullet \cap \bullet t \neq \emptyset$ implies $t'\bullet \subseteq \bullet t$).

Extended SWF-nets have three main properties. The first is that all length-one-loop transitions have all of their input places as output places too. This property assures that length-one-loop transitions can be safely removed during the pre-processing phase. The second property is that if a transition t shares output places with another length-one-loop transition t', then all output places of t are connected to t' too. This property assures the correct detection of all the places to which the length-one-loop transitions should be connected to. The third property is that, when removing all the length-one-loop transitions (and their connecting arcs) from an extended SWF-net, the remaining net is a SWF-net. This property assures the α-algorithm correctly mines the remaining net. As

an example, note that the sound WF-net in Figure 6 is an extended SWF-net, but the sound WF-net in Figure 7 is not.

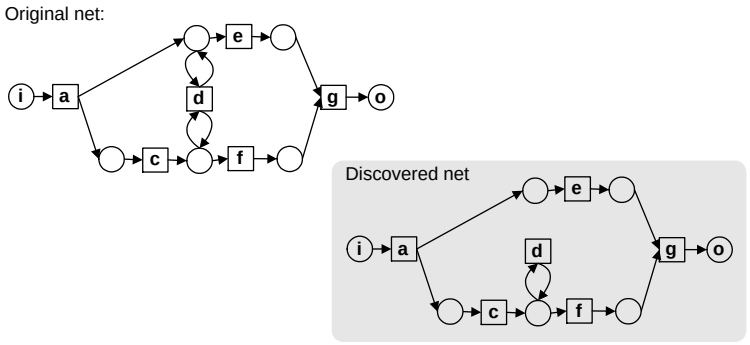

Fig. 7. Example of a sound WF-net that the *modified version* of the α^+-algorithm (the α^{++}) does not correctly mine.

6 Literature on Process Mining

The idea of process mining is not new [7, 9–11, 17–19, 24, 25, 32, 33, 5, 36, 6]. 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. 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 the nature of workflow graphs there is no need to identify the nature (AND or OR) of joins and splits. As shown in [22], 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 [25], a tool based on these algorithms is presented. Schimm [32, 33] has developed a mining tool suitable for discovering hierarchically structured workflow processes. This requires all splits and joins to be balanced. Herbst and Karagiannis also address the issue of process mining in the context of workflow management [18, 17, 19] using an inductive approach. The work presented in [19] is limited to sequential models. The approach described in [18, 17] 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, 25]. 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 [24, 36] is characterized by the focus on workflow processes with concurrent behavior (rather than adding ad-hoc mechanisms to capture parallelism). In [36] a heuristic approach using rather simple metrics is used to construct so-called "dependency/frequency tables" and "dependency/frequency graphs". The preliminary results presented in [36] only provide heuristics and focus on issues such as noise. In [3] the EMiT tool is presented which uses an extended version of the α-algorithm to incorporate timing information. Now EMiT also incorporates the ideas presented in this paper. For a detailed description of the α-algorithm and a proof of its correctness we refer to [6]. For a detailed explanation of the constructs the α-algorithm does not correctly mine see [26].

Process mining can be seen as a tool in the context of Business (Process) Intelligence (BPI). In [16] 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 as data warehouse simplifies and speeds up the queries needed to derive causal relations. In [13, 27, 28] the design of such warehouse and related issues are discussed in the context of workflow logs. Moreover, [28] 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) [20]. The later tool is commercially available and a customized version of PPM is the Staffware Process Monitor (SPM) [34] which is tailored towards mining Staffware logs. Note that none of the latter tools is extracting the process model. The main focus is on clustering and performance analysis rather than causal relations as in [7, 9–11, 17–19, 24, 25, 32, 33, 36].

More from a theoretical point of view, the rediscovery problem discussed in this paper is related to the work discussed in [8, 15, 30]. In these papers the limits of inductive inference are explored. For example, in [15] 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, 15, 30] 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). For a survey of existing research, we also refer to [5].

7 Conclusion

The focus of this paper has been the extension of the α-algorithm so that it can mine all sound SWF-nets. The new algorithm is called α^+. The α-algorithm is proven to correctly discover sound SWF-nets without lenght-one or length-two loops. The extension involved changes in the pre- and post-processing phases. First, length-two loops were tackled by redefining the notion of log completeness and the possible ordering relations among tasks in the process. This solution dealt with the pre-processing phase only. Then, the solution to tackle length-two loops was extended to tackle also length-one loops. The key property is that length-one-loop tasks are connected to single places in sound SWF-nets. Therefore, the α^+-algorithm (i) removes all occurrences of length-one loops from the input log, (ii) feeds in the α-algorithm with this log and the new defined ordering relations over this log, and (iii) reconnects all the length-one loop tasks to their respective place in the net the α-algorithm produced. We proved that the α^+-algorithm correctly mines nets in the class of sound SWF-nets. This new algorithm is implemented in the EMiT tool[7].

In this paper a formal approach has been presented. We presuppose perfect information: (i) the log must be complete (i.e., if a task can follow another

[7] The EMiT tool is available at our research group's website (www.processmining.org).

task directly, the log contains an example of this behavior) and (ii) there is no noise in the log (i.e., everything that is registered in the log is correct). The advantages of a formal approach is that we can *prove* under which conditions our algorithm will certainly discover the right workflow net. However, real logs are rarely complete and/or noise free. For this reason we also try to developed heuristic mining techniques and tools which are less sensitive for noise and the incompleteness of logs.

However, note that even noisy or incomplete logs *not always* make it impossible to use the algorithm α^+ or α. The reason is that real logs usually also contains more data than we assume. This extra data may help in correctly inferring the ordering relations even if the log is incomplete or noisy. Additionally, based on this extra data, the workflow log given as input can also be treated to contain all necessary transitions and loops. As a result, the pre-processing phase is modified, but the processing and post-processing phases remain the same.

In future research we try to develop stronger mining algorithms that can deal with a wider class of workflow nets.

Acknowledgements

A.K.A. de Medeiros would like to thank P.C.W. van den Brand for the helpful discussions about theorem proving.

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 and B.F. van Dongen. Discovering Workflow Performance Models from Timed Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, volume 2480 of *Lecture Notes in Computer Science*, pages 45–63. Springer-Verlag, Berlin, 2002.
4. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
5. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering, Accepted for publication, 2003.
6. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering (TKDE), Accepted for publication, 2003.
7. 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.
8. D. Angluin and C.H. Smith. Inductive Inference: Theory and Methods. *Computing Surveys*, 15(3):237–269, 1983.

9. 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.
10. 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.
11. 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.
12. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
13. J. Eder, G.E. Olivotto, and Wolfgang Gruber. A Data Warehouse for Workflow Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, volume 2480 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, Berlin, 2002.
14. L. Fischer, editor. *Workflow Handbook 2001, Workflow Management Coalition*. Future Strategies, Lighthouse Point, Florida, 2001.
15. E.M. Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3):302–320, 1978.
16. D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168. Morgan Kaufmann, 2001.
17. 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.
18. J. Herbst. *Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen*. PhD thesis, Universität Ulm, November 2001.
19. 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.
20. IDS Scheer. ARIS Process Performance Manager (ARIS PPM). http://www.ids-scheer.com, 2002.
21. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
22. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows (submitted)*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2002. Available via http://www.tm.tue.nl/it/research/patterns.
23. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
24. 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. In *Proceedings of the 5th International Conference on Discovery Science (Discovery Science 2002)*, volume 2534 of *Lecture Notes in Artificial Intelligence*, pages 364–373. Springer-Verlag, Berlin, 2002.

25. 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.
26. A.K.A. de Medeiros, W.M.P. van der Aalst, and A.J.M.M. Weijters. Workflow mining: Current status and future directions. In Robert Meersman, Zahir Tari, and Douglas C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *LNCS*, pages 389–406. Springer-Verlag, 2003.
27. M. zur Mühlen. Process-driven Management Information Systems Combining Data Warehouses and Workflow Technology. In B. Gavish, editor, *Proceedings of the International Conference on Electronic Commerce Research (ICECR-4)*, pages 550–566. IEEE Computer Society Press, Los Alamitos, California, 2001.
28. M. zur Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33)*, pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.
29. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
30. L. Pitt. Inductive Inference, DFAs, and Computational Complexity. In K.P. Jantke, editor, *Proceedings of International Workshop on Analogical and Inductive Inference (AII)*, volume 397 of *Lecture Notes in Computer Science*, pages 18–44. Springer-Verlag, Berlin, 1889.
31. 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.
32. G. Schimm. Process Mining. http://www.processmining.de/.
33. G. Schimm. Process Miner - A Tool for Mining Process Schemes from Event-based Data. In S. Flesca and G. Ianni, editors, *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA)*, volume 2424 of *Lecture Notes in Computer Science*, pages 525–528. Springer-Verlag, Berlin, 2002.
34. Staffware. Staffware Process Monitor (SPM). http://www.staffware.com, 2002.
35. 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.
36. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.