Process Mining for Ubiquitous Mobile Systems: An Overview and a Concrete Algorithm

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. Ubiquitous Mobile Systems (UMSs) allow for automated capturing of events. Both mobility and ubiquity are supported by electronic means such as mobile phones and PDAs and technologies such as RFID, Bluetooth, WLAN, etc. These can be used to automatically record human behavior and business processes in detail. UMSs typically also allow for more flexibility. The combination of flexibility (i.e., the ability to deviate from standard procedures) and the automated capturing of events, provides an interesting application domain for *process mining*. The goal of process mining is to discover process models from event logs. The α-algorithm is a process mining algorithm whose application is not limited to ubiquitous and/or mobile systems. Unfortunately, the α-algorithm is unable to tackle so-called "short loops", i.e., the repeated occurrence of the same event. Therefore, a new algorithm is proposed to deal with short loops: the α^+-algorithm. This algorithm has been implemented in the EMiT tool.

Keywords: process mining, workflow mining, Petri nets, mobile computing, ubiquitous computing.

1 Process Mining in Ubiquitous Mobile Systems

Since Mark Weiser first described it in 1991 [19], ubiquitous computing has transformed from a fairy-tale to reality. Unlike the traditional situation where people share a computer (mainframe) or where there is a one-to-one correspondence between people and computers (PC), people are surrounded by computing devices. These devices may have a fixed location (cf. "smart homes"), but increasingly these devices are mobile. Typical examples that exist today are PDA's and mobile phones. New concepts like "smart clothes" or "wearable computers" illustrate future applications. Research projects such as UWA (Ubiquitous Web Applications, cf. www.uwaproject.org) and MOTION (MObile Teamwork Infrastructure for Organizations Networks, cf. www.motion.softeco.it) illustrate the scientific challenges ubiquitous and/or mobile systems are posing.

Clearly, Ubiquitous Mobile Systems (UMSs) will change the way people work [10]. However, in this paper we do not discuss the impact of new technologies on people and organizations but focus on the use of process mining to monitor the processes where UMSs are involved. The main contribution is an extension of the α-algorithm [7] to tackle so-called "short loops" [13], i.e., the paper consists of two clearly distinguishable parts: (1) an introduction to process mining and its relevance for UMSs and (2) a concrete algorithm called the α^+-algorithm. We start with the first part and discuss the relation between process mining and UMSs.

UMSs are enabled by recent developments in various fields. Miniaturization and mass fabrication of "computer-like devices" allows for an omnipresence of computing power. This combined with advances in wireless communication have made mobile computing a reality. The difference between phones and portable computers (e.g., PDAs) is fading, thus illustrating this trend. Current technologies such as Bluetooth and WLAN enable new ways of cooperation. RFID tags allows for new ways of "synchronizing" the physical and electronic worlds. These technologies have in common that potentially it can be used to automatically capture data on work processes and other human behavior. Humans are very good in problem solving but are typically not very accurate or thorough in recording information on events that take place. A human's sensory and short term memories have limited capacity and data entry is considered a tedious job. Therefore, UMSs can serve as natural assistants to record events.[1]

UMSs typically allow for more flexibility than traditional information systems. For many work processes this increased flexibility will trigger the need to monitor things more closely. The combination of availability of event data and the desire to monitor suggests the use of *process mining* techniques. Fueled by the omnipresence of event logs in transactional information systems (cf. WFM, ERP, CRM, SCM, and B2B systems), process mining has become a vivid research area [5, 6]. Until recently, the information in these event logs was rarely used to analyze the underlying processes. Process mining aims at improving this by providing techniques and tools for discovering process, control, data, organizational, and social structures from event logs, i.e., the basic idea of process mining is to diagnose business processes by mining event logs for knowledge.

The event log typically contains information about events referring to an *activity* and a *case*. The case (also named process instance) is the "thing" which is being handled, e.g., a customer order, a job application, an insurance claim, a building permit, etc. The activity (also named task, operation, action, or work-item) is some operation on the case. Typically, events have a *timestamp* indicating the time of occurrence. Moreover, when people are involved, event logs will typically contain information on the person executing or initiating the event, i.e., the *originator*. Based on this information several tools and techniques for process mining have been developed [2, 4, 5, 7–9, 11, 12, 15, 17, 18]. In this paper we present the α^+-algorithm. This is a new algorithm focusing on the control-flow (i.e., process) perspective. It extends the α-algorithm [7] by addressing the problem of "short loops" [13].

The remainder of this paper is organized as follows. Section 2 introduces the concept of process mining. Section 3 describes the α-algorithm and its supporting

[1] Note that in most mobile systems there is a communication asymmetry, i.e., the bandwidth downstream (server-to-client) is much larger than upstream. This may complicate data collection in a mobile system. However, the bandwidth required to record events is typically moderate.

definitions. Section 4 presents the new approach to tackle length-two loops using the α-algorithm. Section 5 shows how to extend the approach in Section 4 to mine also length-one loops. Section 6 discusses related works. Section 7 has the conclusions.

2 Process Mining: An Introduction

We assume that in UMSs, information on events (e.g., the execution of a task by a worker) is recorded in a log.

To illustrate the principle of process mining, we consider the event 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 an event 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 [16]. 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.

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. An event log.

For this simple example, it is quite easy to construct a process model that is able to regenerate the event 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 [18]. These are just some of the problems that we need to face in process mining research. In this paper 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).

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 between tasks are inferred. The processing phase corresponds to the execution of

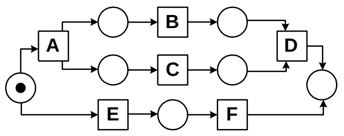

Fig. 1. A process model corresponding to the event log.

the mining algorithm, given the log and the ordering relations as input. In our case, the mining algorithm is the α-algorithm [7]. 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 event log. Considerable work has been done on heuristics to mine event-data logs to produce a process model. Typically these models can be characterized as workflow models. Existing heuristic-based mining algorithms have limitations as indicated in [13]. Typically, more advanced process constructs are difficult to handle for existing mining algorithms. Some of these problematic constructs are common in workflow applications and, therefore, need to be addressed to enable application in practice. Among these constructs are short loops (see Figure 2) .

The main aim of our research is to extend the class of nets we can correctly mine. The α-algorithm is able to correctly mine sound SWF-nets without short loops [7]. 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.

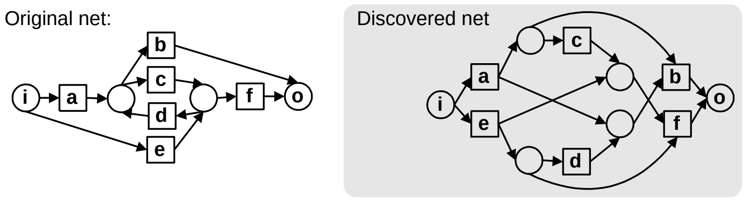

Fig. 2. An example of a sound SWF-net the α-algorithm cannot correctly mine.

3 WF-nets and the α-Algorithm

This section contains the main definitions used in the α-algorithm that are also relevant to the new α^+-algorithm presented in this paper. For more information on the α-algorithm and Structured Workflow Nets (SWF-nets) the reader is referred to [7]. We assume some basic knowledge of Petri nets. Readers not familiar with basic concepts such as (P, T, F) as a representation for a Petri net, the firing rule, firing sequences, preset $\bullet x$, postset $x\bullet$, boundedness, liveness, reachability, etc. are referred to [1, 16].

3.1 Workflow Nets

Before introducing the α-algorithm we briefly discuss a subclass of Petri nets called a *WorkFlow nets* (WF-nets). This subclass is tailored towards modeling the control-flow dimension of a workflow.[2] It should be noted that a WF-net specifies the dynamic behavior of a single case in isolation [1].

Definition 3.1. (Workflow nets) Let $N = (P, T, F)$ be a Petri 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 Petri 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 3.1, 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 3.2. (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 [1, 3].

Most workflow systems offer standard building blocks such as the AND-split, AND-join, XOR-split, and XOR-join [3]. These are used to model sequential, conditional, parallel and iterative routing. Clearly, a WF-net can be used to specify the routing of cases. *Tasks*, also referred to as *activities*, 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. XOR-splits/XOR-joins correspond to places with multiple outgoing/ingoing arcs. Given the close relation between tasks and transitions we use the terms interchangeably.

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.

[2] Note that we use the words *workflow* and *process* interchangeably.

Definition 3.3. (Implicit place) Let $N = (P, T, F)$ be a Petri 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 $t \in p\bullet$, $s' \geq \bullet t \setminus \{p\} \Rightarrow s' \geq \bullet t$.[3]

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.

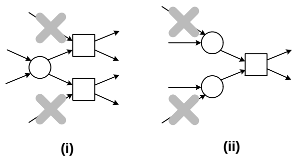

Fig. 3. Constructs not allowed in SWF-nets.

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 XOR-join. WF-nets which satisfy these requirements are named *structured workflow nets* and are defined as:

Definition 3.4. (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 4) and then also length-one loops (see Section 5). 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 3.5. (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 $t \in T$, $t\bullet \cap \bullet t = \emptyset$.

[3] $[N, s\rangle$ is the set of reachable markings of net N when starting in marking s, $p\bullet$ is the set of output transitions of p, $\bullet t$ is the set of input places of t, and $\geq$ is the standard ordering relation on multisets.

3.2 The α-Algorithm

The starting point for process mining is the event log. A log is a set of traces. Event traces and logs are defined as:

Definition 3.6. (Event trace, event log) Let T be a set of tasks. $\sigma \in T^*$ is an *event trace* and $W \in \mathcal{P}(T^*)$ is an *event log.*[4]

From an event log, ordering relations between tasks can be inferred. In the case of the α-algorithm, every two tasks in the event 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 event traces. The ordering relations are defined as:

Definition 3.7. (Log-based ordering relations) Let W be an event 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 event log contains the minimal amount of information necessary to mine the process, the notion of log completeness is defined as:

Definition 3.8. (Complete event log) Let $N = (P, T, F)$ be a sound WF-net, i.e., $N \in \mathcal{W}$. W is an *event 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* event log of N if and only if (1) for any event log W' of N: $>_{W'} \subseteq >_W$, and (2) for any $t \in T$ there is a $\sigma \in W$ such that $t \in \sigma$.

For Figure 1, a possible complete event 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 $c \|_W b$.

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

Definition 3.9. (Mining algorithm α) Let W be an event 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. $T_I = \{t \in T \mid \exists_{\sigma \in W} t = \mathit{first}(\sigma)\}$,
3. $T_O = \{t \in T \mid \exists_{\sigma \in W} t = \mathit{last}(\sigma)\}$,

[4] 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., $W \subseteq T^*$.

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 event 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[5]. In steps 4 and 5, the α-algorithm creates sets (X_W and Y_W, respectively) used to define the places of the discovered WF-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 XOR-split/join constructs. Note that the XOR-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 WF-net is returned in Step 8.

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

Definition 3.10. (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 event logs of N onto sound WF-nets, i.e., $\alpha : \mathcal{P}(T^*) \rightarrow \mathcal{W}$. If for any complete event 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.

4 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.

[5] In a WF-net, the source place i has no input transitions and the sink place o has no output transitions.

Log completeness as defined in Definition 3.8 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 whether 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 4.1. (Loop-complete event log) Let $N = (P, T, F)$ be a SWF-net and W a log of N. W is a *loop-complete event log of* N if and only if W is complete and for all event logs W' of N: if there is a firing sequence $\sigma' \in W'$ with $\sigma' = t_1 t_2 t_3 \ldots t_{n'}$ and $i' \in \{1, \ldots, n' - 2\}$ 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 $\sigma = t_1 t_2 t_3 \ldots t_n$ and $i \in \{1, \ldots, n - 2\}$ such that $t_i = t_{i+2} = a$ and $t_{i+1} = b$.

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

The new notion of a loop-complete event 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 event 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 3.7, i.e., we provide the following new definitions for the basic ordering relations $\rightarrow_W$ and $\|_W$.

Definition 4.2. (Ordering relations capturing length-two loops) Let W be a loop-complete event 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 $\sigma = t_1 t_2 t_3 \ldots t_n$ and $i \in \{1, \ldots, n - 2\}$ 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 $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ 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 $(b \not>_W a$ 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 4.2, 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 event traces.

However, there is still a problem. Length-one loops in the net may also produce "*cdc*" and "*dcd*" patterns in the event traces, cf. Figure 4. Therefore, to prove that the α-algorithm can correctly mine length-two loops when using the new definitions of loop-complete event log and ordering relations, we assume that the net is a *sound one-loop-free SWF-net*.

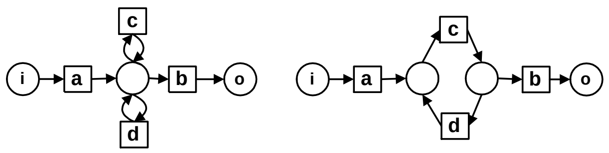

Fig. 4. Example illustrating why length-one loops are not allowed when mining length-two loops. Note that both nets have loop-complete event logs that contain traces with the substrings "*cdc*" or "*dcd*".

Theorem 4.3. Let $N = (P, T, F)$ be an one-loop-free sound SWF-net. Let W be a loop-complete event 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. For a detailed proof we refer to [14]. $\Box$

Using this result, it is possible to prove the following theorem which shows that the new ordeing relations solve the problem of length-two loops.

Theorem 4.4. Let $N = (P, T, F)$ be a sound one-loop-free SWF-net and let W be a loop-complete event log of N. Then $\alpha(W) = N$ modulo renaming of places.

Proof. See [14] for a detailed proof. $\Box$

In the next section, we show how to handle the general case (all sound SWF-nets).

5 Length-One Loops

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

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

Theorem 5.1. 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. See [14]. $\Box$

Property 5.2. Let $N = (P, T, F)$ be a sound SWF-net. Let W be a complete event 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 $\bullet c = \bullet a$.

Theorem 5.3. 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 event log of N and let W^{-L1L} be the log created by excluding the occurrences of length-one-loop transitions from every event trace in W. Then:

1. N' is a sound one-loop-free SWF-net,

2. $\alpha(W^{-L1L}) = N'$ modulo renaming of places.

Proof. See [14]. □

Theorem 5.3 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 5.1), 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 solution to tackle length-one loops in sound SWF-nets focuses 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 event 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). Figure 5 shows the basic idea by illustrating a pre- and post-processing phase.

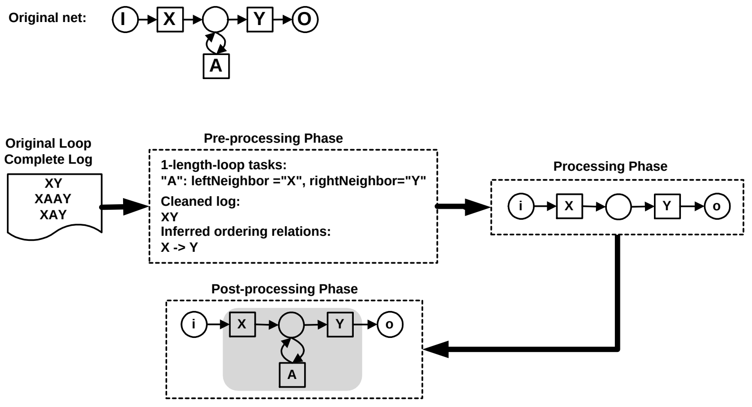

Fig. 5. Example of the approach to tackle length-one loops in sound SWF-net.

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

Definition 5.4. (Mining algorithm α^+) Let W be a loop-complete event log over T, the α-algorithm as in Definition 3.9 and the ordering relations as in Definition 4.2.

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. $\sigma' := \mathit{eliminateTask}(\sigma', t)$
   (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 event 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 event log W^{-L1L} and the ordering relations as defined in Definition 4.2. 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 5.5. Let $N = (P, T, F)$ be a sound SWF-net and let W be a loop-complete event log of N. Using the ordering relations as in Definition 4.2, $\alpha^+(W) = N$ modulo renaming of places.

Proof. See [14]. □

The original net in Figure 2 and the nets N_{1-4} in Figure 6 satisfy the requirements stated in Theorem 5.5. 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 (cf. [14]).

6 Related Work

The idea of process mining is not new [2, 5, 7–9, 11–13, 15, 17, 18] and most techniques aim at the control-flow perspective. However, process mining is not limited

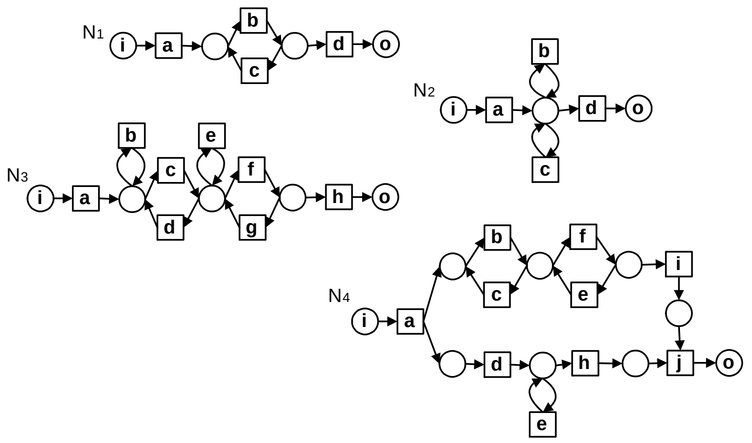

Fig. 6. Examples of sound SWF-nets that the α^+-algorithm correctly mines.

to the control-flow perspective. For example, in [4] we use process mining techniques to construct a social network. For more information on process mining we refer to a special issue of Computers in Industry on process mining [6] and a survey paper [5]. In this paper, unfortunately, it is impossible to do justice to the work done in this area.

The focus of this paper is on an extension of the α-algorithm. For more information on the α-algorithm, we refer to [2, 7, 13, 18]. This paper tackles one of the problems raised in [13] and should be considered as an extension of [7]. For more detailed proofs we refer to a technical report [14].

To support our mining efforts we have developed a set of tools including EMiT [2], Thumb [18], and MinSoN [4]. These tools share a common XML format. For more details we refer to www.processmining.org.

7 Conclusion

New technologies such as RFID, Bluetooth, WLAN, etc. and the omnipresence of small computing devices (e.g., mobile phones and PDAs) enable UMSs that can be used to record human behavior and business processes in detail. The combination of flexibility (i.e., the ability to deviate from standard procedures) and the automated capturing of events, suggests that UMSs form an interesting application domain for process mining. In this paper, we did not elaborate on the application of process mining in this domain. Instead, we focused on an extension of the α-algorithm such 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 length-one or length-two loops. The extension involves changes in the pre- and post-processing phases. First, length-two loops are tackled by redefining the notion of log completeness and the possible ordering relations among tasks in the process. This solution is addressed in the pre-processing phase only. Then, the solution to tackle length-two loops is 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 (1) removes all occurrences of length-one loops from the input log, (2) feeds in the α-algorithm with this log and the new defined ordering relations, and (3) reconnects all the length-one loop tasks to their respective place in the net the α-algorithm produced. In this paper and a technical report [14], we provide a formal proof that the α^+-algorithm correctly mines nets in the *whole* class of sound SWF-nets.

Through our website, www.processmining.org, we provide the tool EMiT that implements the α^+-algorithm. Note that EMiT uses a generic XML-based input format. In addition, EMiT provides adapters for tools like Staffware, InConcert, ARIS PPM, etc. We invite people developing UMSs to log information in our XML format (cf. www.processmining.org). Examples of possible applications we envision include:

– Patients and medical equipment are tagged while the medical staff is equipped with small computing devices. This way health-care processes can be monitored to audit medical protocols, support the medical staff (e.g, alerts), and suggest improvements.
– Service engineers are equipped with PDAs, mobile phones (for internet connection), and GPS (for location). This way service processes can be analyzed.
– Students use ubiquitous/mobile technologies to study anytime/anywhere. To monitor progress and act accordingly, process mining is used to analyze the learning process.

Note that in each of these applications there will be repetitions. Therefore, it is essential that the α^+-algorithm is able to tackle short loops.

Acknowledgements

The authors would like to thank Minseok Song, Laura Maruster, Eric Verbeek, Monique Jansen-Vullers, Hajo Reijers, Michael Rosemann, and Peter van den Brand for their joint efforts on process mining techniques and tools at Eindhoven University of Technology.

References

1. 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.
2. 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.
3. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2002.
4. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering interaction patterns in business processes. In M. Weske, B. Pernici, and J. Desel, editors, *International Conference on Business Process Management (BPM 2004)*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2004.

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*, 47(2):237–267, 2003.
6. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
7. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. QUT Technical report, FIT-TR-2003-03, Queensland University of Technology, Brisbane, 2003. (Accepted for publication in IEEE Transactions on Knowledge and Data Engineering.).
8. 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.
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. S. Dustdar, H. Gall, and R. Schmidt. Web Services For Groupware in Distributed and Mobile Collaboration. In P. Cremonesi, editor, *Proceeding of the 12th IEEE Euromicro Conference on Parallel, Distributed and Network based Processing (PDP 2004)*, pages 241–247. IEEE Computer Society, Los Alamitos, CA, 2004.
11. 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.
12. IDS Scheer. ARIS Process Performance Manager (ARIS PPM). http://www.ids-scheer.com, 2002.
13. 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 R. Meersman, Z. Tari, and D.C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 389–406. Springer-Verlag, Berlin, 2003.
14. A.K.A. de Medeiros, B.F. van Dongen, W.M.P. van der Aalst, and A.J.M.M. Weijters. Process Mining: Extending the α-algorithm to Mine Short Loops. BETA Working Paper Series, WP 113, Eindhoven University of Technology, Eindhoven, 2004.
15. 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.
16. 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.
17. M. Sayal, F. Casati, and M.C. Shan U. Dayal. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.
18. 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.
19. M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.